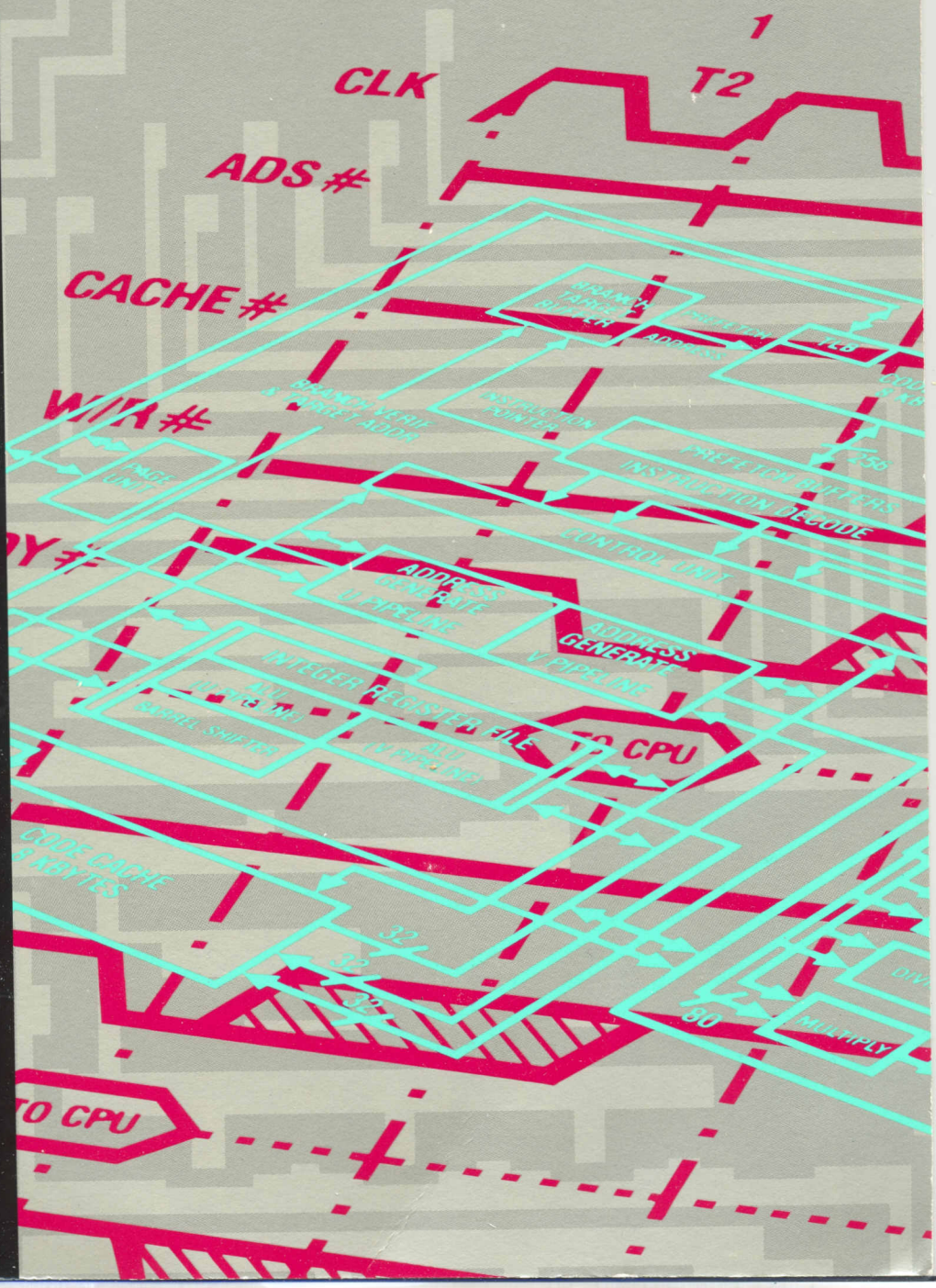
Microprocessors: Volume II

Intel486™ Microprocessors



intel

LITERATURE

To order Intel literature or obtain literature pricing information in the U.S. and Canada call or write Intel Literature Sales. In Europe and other international locations, please contact your ***local*** sales office or distributor.

INTEL LITERATURE SALES
P.O. Box 7641
Mt. Prospect, IL 60056-7641

In the U.S. and Canada
call toll free
(800) 548-4725
This 800 number is for external customers only.

CURRENT HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information. All handbooks can be ordered individually, and most are available in a pre-packaged set in the U.S. and Canada.

Title	Intel Order Number	ISBN
SET OF FOURTEEN HANDBOOKS (Available in U.S. and Canada)	**231003**	**N/A**
CONTENTS LISTED BELOW FOR INDIVIDUAL ORDERING:		
CONNECTIVITY	231658	1-55512-202-7
EMBEDDED MICROCONTROLLERS	270646	1-55512-203-5
EMBEDDED MICROPROCESSORS	272396	1-55512-204-3
FLASH MEMORY (2 volume set)	210830	1-55512-214-0
MICROPROCESSORS, VOL. 1: Intel386™ 80286 & 8086 MICROPROCESSORS	230843	1-55512-196-9
MICROPROCESSORS, VOL. 2: Intel486™ MICROPROCESSORS	241731	1-55512-197-7
MICROPROCESSORS, VOL. 3: PENTIUM™ PROCESSORS	241732	1-55512-198-5
i750®, i860™, i960® PROCESSORS AND RELATED PRODUCTS	272084	1-55512-217-5
OEM BOARDS, SYSTEMS & SOFTWARE	280407	1-55512-201-9
PACKAGING	240800	1-55512-208-6
PERIPHERAL COMPONENTS	296467	1-55512-207-8
PRODUCT OVERVIEW	210846	N/A
PROGRAMMABLE LOGIC	296083	1-55512-206-X
NETWORKING	297360	1-55512-220-5
ADDITIONAL LITERATURE: (Not included in handbook set)		
AUTOMOTIVE PRODUCTS	231792	1-55512-212-4
COMPONENTS QUALITY/RELIABILITY	210997	1-55512-132-2
CUSTOMER LITERATURE GUIDE	210620	N/A
EMBEDDED APPLICATIONS (1993/94)	270648	1-55512-179-9
INTERNATIONAL LITERATURE GUIDE (Available in Europe only)	E00029	N/A
MILITARY AND SPECIAL PRODUCTS (2 volume set)	210461	1-55512-213-2
SYSTEMS QUALITY/RELIABILITY	231762	1-55512-046-6

Microprocessors: Volume II

In 1993, Intel enhanced its entire Intel486™ microprocessor family with energy-efficient technology. The SL Enhanced Intel486 microprocessor family enables built-in power management while maintaining full compatibility with existing Intel architecture processors and over $50 billion of software packages. The SL Enhanced Intel486 CPU family allows system designers to design desktop systems that exceed the Environmental Protection Agency's (EPA) EnergyStar guidelines, without sacrificing performance. Bringing SL technology to the entire Intel486 CPU line increases notebook system design flexibility and increases the battery life of all high-performance Intel486 CPU-based notebooks. By providing multiple performance options, Intel makes desktop and mobile computer power affordable for more and more users.

This handbook contains extensive information on Intel486 microprocessor families, numeric coprocessors, cache and memory controllers, floppy and hard disk controllers, mobile peripheral products, flash memory components, PCMCIA cards for the desktop and mobile family of Intel486 microprocessors.

The data sheets and application notes contained in this handbook contain comprehensive charts, diagrams, instructions, and hardware information for leading-edge 32-bit system development.

intel®

Order Number: 241731-001
Printed in USA/194/25K/RRD/JW
Microprocessors

ISBN 1-55512-197-7

90000

9 781555 121976

Printed on Recycled Material

intel®

MICROPROCESSORS

VOLUME II

1994

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

MDS is an ordering code only and is not used as a product name or trademark of Intel Corporation.

Intel Corporation and Intel's FASTPATH are not affiliated with Kinetics, a division of Excelan, Inc. or its FASTPATH trademark or products.

*Other brands and names are the property of their respective owners.

Additional copies of this document or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

©INTEL CORPORATION, 1993

LGCPY1/100693

DATA SHEET DESIGNATIONS

Intel uses various data sheet markings to designate each phase of the document as it relates to the product. The marking appears in the upper, right-hand corner of the data sheet. The following is the definition of these markings:

Data Sheet Marking	Description
Product Preview	Contains information on products in the design phase of development. Do not finalize a design with this information. Revised information will be published when the product becomes available.
Advanced Information	Contains information on products being sampled or in the initial production phase of development.*
Preliminary	Contains preliminary information on new products in production.*
No Marking	Contains information on products in full production.*

*Specifications within these data sheets are subject to change without notice. Verify with your local Intel sales office that you have the latest data sheet before finalizing a design.

intel

DATA SHEET DESIGNATIONS

Intel uses various data sheet markings to designate each phase of the document as it relates to the product. The marking appears in the upper, right-hand corner of the data sheet. The following is the definition of these markings:

Data Sheet Marking	Description
Product Preview	Contains information on products in the design phase of development. Do not finalize a design with this information. Revised information will be published when the product becomes available.
Advanced Information	Contains information on products being sampled or in the initial production phase of development.*
Preliminary	Contains preliminary information on new products in production.*
No Marking	Contains information on products in full production.*

*Specifications within these data sheets are subject to change without notice. Verify with your local Intel sales office that you have the latest data sheet before finalizing a design.

Overview 1

Intel486™ Microprocessor 2

Intel OverDrive™ Processors 3

Peripheral Components 4

Flash Memory Components 5

1 Overview

2 Intel486™ Microprocessor

3 Intel OverDrive™ Processors

4 Peripheral Components

5 Flash Memory Components

Table of Contents

Alphanumeric Index xi

CHAPTER 1
Overview
Introduction Microprocessors Vol II 1-1

CHAPTER 2
Intel486™ Microprocessor
DATA SHEETS
Introduction to the Intel486 Microprocessor Family 2-1
Intel486 DX2 Microprocessor 2-2
Intel486 DX Microprocessor 2-211
Intel486 SX Microprocessor 2-445
SL Enhanced Intel486 Microprocessor Data Sheet Addendum 2-715
3.3V Intel486 SX Microprocessor 2-808
Intel486 Family of Microprocessors Low Power Version 2-816
82420 PCIset 2-852
APPLICATION NOTES
AP-447 A Memory Subsystem for the Intel486 Family of Microprocessors Including Second Level Cache 2-863
AP-453 Clock Design in 50 MHz Intel486 Systems 2-931
AP-458 Designing a Memory Bus Controller for a 50 MHz Intel486 DX CPU-Cache System 2-951
AP-460 Designing a Memory Bus Controller for a 50 MHz Intel486 DX CPU-Cache Write-Through EISA System 2-1004
AP-469 Cache and Memory Design Considerations for the Intel486 DX2 Microprocessor 2-1005
AP-496 Migrating from the Intel486 SL Microprocessor to the SL Enhanced Intel486 Microprocessor 2-1042
AP-497 Managing Power with the SL Enhanced Intel486 Microprocessor 2-1054
AP-498 Thermal Design for High Performance Notebooks 2-1062

CHAPTER 3
Intel OverDrive™ Processors
Intel OverDrive Processors 3-1

CHAPTER 4
Peripheral Components
DESKTOP AND MOBILE PERIPHERAL DATA SHEETS
82091AA Advanced Integrated Peripheral (AIP) 4-1
82078 CHMOS Single-Chip Floppy Disk Controller 4-3
82078 44 Pin CHMOS Single-Chip Floppy Disk Controller 4-6
82078 64 Pin CHMOS Single-Chip Floppy Disk Controller 4-7
82077SL CHMOS Single-Chip Floppy Disk Controller 4-8
89C024FT V.42/42bis Modem Chip Set 4-9
89C024LT Error Correcting Laptop Modem Chip Set 4-10
89C124FX Data/Fax Modem Chip Set 4-11
82595 ISA/PCMCIA High Integration Ethernet Controller 4-12
82593 CSMA/CD Core LAN Controller 4-13
82503 Dual Serial Transceiver (DST) 4-14
Ethernet LAN Card Product Brief 4-15
DataFax 14.4 Card Product Brief 4-18
Faxmodem 24/96 Card Product Brief 4-20
85C220/85C224-100, -80 and -66 Fast Registered Speed Tsu, Tso 8-Macrocell PLDs 4-22
iPLD22V10L-5 Low Power, Electrically Erasable 10-Macrocell CMOS PLD 4-23

Table of Contents (Continued)

iPLDLV22V10-5 Low Voltage, Electrically Erasable 10-Macrocell CMOS PLD 4-24
DESKTOP PERIPHERAL DATA SHEETS
82489DX Advanced Programmable Interrupt Controller (APIC) 4-25
UPI-41AH/42AH Universal Peripheral Interface 8-Bit Slave Microcontroller 4-108
UPI-C42/UPI-L42 Universal Peripheral Interface CHMOS 8-Bit Slave Microcontroller 4-109
MOBILE PERIPHERAL DATA SHEETS
8XC51SL/Low Voltage 8XC51SL Keyboard Controller 4-110
iPLD610 Fast 16-Macrocell CMOS PLD 4-111
iPLD910 Fast 24-Macrocell CMOS PLD 4-112
iPLD22V10-10, -15, -20 High Performance 10-Macrocell CMOS PLD 4-113
iFX740 10 ns FLEXlogic FPGA with SRAM Option 4-114
APPLICATION NOTES
AP-366 89C124FX Data/Fax Modem Chip Set-Reduction of Power Consumption .. 4-115
AP-358 Intel 82077SL for Super Dense Floppies 4-119

CHAPTER 5
Flash Memory Components
DATA SHEETS
28F001BX-T/28F001BX-B 1M (128K x 8) CMOS Flash Memory 5-1
28F200BX-T/B, 28F002BX-T/B 2 Mbit (128K x 16, 256K x 8) Boot Block Flash Memory Family 5-2
28F200BX-TL/BL, 28F002BX-TL/BL 2 Mbit (128K x 16, 256K x 8) Low Power Boot Block Flash Memory Family 5-3
28F400BX-T/B, 28F004BX-T/B 4 Mbit (256K x 16, 512K x 8) Boot Block Flash Memory Family 5-4
28F008SA 8 Mbit (1 Mbit x 8) Flash Memory 5-5
28F016SA 16 Mbit (1 Mbit x 16, 2 Mbit x 8) FlashFile Memory 5-6

Alphanumeric Index

28F001BX-T/28F001BX-B 1M (128K x 8) CMOS Flash Memory 5-1
28F008SA 8 Mbit (1 Mbit x 8) Flash Memory 5-5
28F016SA 16 Mbit (1 Mbit x 16, 2 Mbit x 8) FlashFile Memory 5-6
28F200BX-T/B, 28F002BX-T/B 2 Mbit (128K x 16, 256K x 8) Boot Block Flash Memory Family 5-2
28F200BX-TL/BL, 28F002BX-TL/BL 2 Mbit (128K x 16, 256K x 8) Low Power Boot Block Flash Memory Family 5-3
28F400BX-T/B, 28F004BX-T/B 4 Mbit (256K x 16, 512K x 8) Boot Block Flash Memory Family 5-4
3.3V Intel486 SX Microprocessor 2-808
82077SL CHMOS Single-Chip Floppy Disk Controller 4-8
82078 44 Pin CHMOS Single-Chip Floppy Disk Controller 4-6
82078 64 Pin CHMOS Single-Chip Floppy Disk Controller 4-7
82078 CHMOS Single-Chip Floppy Disk Controller 4-3
82091AA Advanced Integrated Peripheral (AIP) 4-1
82420 PCIset 2-852
82489DX Advanced Programmable Interrupt Controller (APIC) 4-25
82503 Dual Serial Transceiver (DST) 4-14
82593 CSMA/CD Core LAN Controller 4-13
82595 ISA/PCMCIA High Integration Ethernet Controller 4-12
85C220/85C224-100, -80 and -66 Fast Registered Speed Tsu, Tso 8-Macrocell PLDs 4-22
89C024FT V.42/42bis Modem Chip Set 4-9
89C024LT Error Correcting Laptop Modem Chip Set 4-10
89C124FX Data/Fax Modem Chip Set 4-11
8XC51SL/Low Voltage 8XC51SL Keyboard Controller 4-110
AP-358 Intel 82077SL for Super Dense Floppies 4-119
AP-366 89C124FX Data/Fax Modem Chip Set-Reduction of Power Consumption 4-115
AP-447 A Memory Subsystem for the Intel486 Family of Microprocessors Including Second Level Cache 2-863
AP-453 Clock Design in 50 MHz Intel486 Systems 2-931
AP-458 Designing a Memory Bus Controller for a 50 MHz Intel486 DX CPU-Cache System 2-951
AP-460 Designing a Memory Bus Controller for a 50 MHz Intel486 DX CPU-Cache Write-Through EISA System 2-1004
AP-469 Cache and Memory Design Considerations for the Intel486 DX2 Microprocessor .. 2-1005
AP-496 Migrating from the Intel486 SL Microprocessor to the SL Enhanced Intel486 Microprocessor 2-1042
AP-497 Managing Power with the SL Enhanced Intel486 Microprocessor 2-1054
AP-498 Thermal Design for High Performance Notebooks 2-1062
DataFax 14.4 Card Product Brief 4-18
Ethernet LAN Card Product Brief 4-15
Faxmodem 24/96 Card Product Brief 4-20
iFX740 10 ns FLEXlogic FPGA with SRAM Option 4-114
Intel OverDrive Processors 3-1
Intel486 DX Microprocessor 2-211
Intel486 DX2 Microprocessor 2-2
Intel486 Family of Microprocessors Low Power Version 2-816
Intel486 SX Microprocessor 2-445
Introduction Microprocessors Vol II 1-1
Introduction to the Intel486 Microprocessor Family 2-1
iPLD22V10-10, -15, -20 High Performance 10-Macrocell CMOS PLD 4-113
iPLD22V10L-5 Low Power, Electrically Erasable 10-Macrocell CMOS PLD 4-23
iPLD610 Fast 16-Macrocell CMOS PLD 4-111
iPLD910 Fast 24-Macrocell CMOS PLD 4-112

SL Enhanced Intel486 Microprocessor Data Sheet Addendum 2-715
UPI-41AH/42AH Universal Peripheral Interface 8-Bit Slave Microcontroller 4-108
UPI-C42/UPI-L42 Universal Peripheral Interface CHMOS 8-Bit Slave Microcontroller 4-109

Overview

1

1 Overview

INTRODUCTION

Intel microprocessors and peripherals provide a broad range of time-saving, energy-efficient, high-performance solutions to designers of both mobile and desktop microprocessor-based systems. Intel's microprocessor/peripheral interface delivers *time* and *performance* advantages to the designers of microprocessor-based systems, meeting their demand for greater performance, lower power consumption and a wider variety of built-in features for their customers.

IMPROVED SYSTEM PERFORMANCE

Intel offers an entire product line of microprocessors for desktop and mobile systems, ranging from the 25 MHz version of the Intel486™ SX processor to the high performance 66 MHz Intel486 DX2 processor. As Intel maintains its leadership in MIPS for microprocessor-based systems, it has managed to couple this superior performance with its sophisticated energy-efficient SL technology to meet the requirements of the EPA's Energy Star guidelines.

Intel has combined SL technology with desktop performance to enhance its entire Intel486 microprocessor family. Now, with built-in power management, SL Enhanced Intel486 CPU-based mobile systems achieve increases in battery life without compromising performance, while maintaining full compatibility with existing Intel architecture processors. SL Enhanced Intel486 CPUs for mobile and desktop systems are available in a full range of speeds (25 MHz to 66 MHz), packages (PGA, SQFP, PQFP), and voltages (5V, 3.3V) to meet any system design requirement.

REDUCED TIME TO MARKET

Intel's universal motherboard for Intel486 SX, DX, and DX2 microprocessor-based desktop systems and scalable architecture for mobile systems reduces the development effort required to produce an entire product line of high-performance systems, thereby reducing time to market.

Intel offers a wide variety of off-the-shelf components to fulfill the requirements of system designers while simplifying the implementation of their designs. Off-the-shelf system solutions greatly decrease the potential risk of costly project delays due to component incompatibility. These system solutions greatly reduce the amount of time required to design, debug, manufacture and test microprocessor-based systems.

INCREASED RELIABILITY

At Intel, the rate of failure for devices is carefully tracked. Highest reliability is a tangible goal that translates to higher reliability for your product, reduced downtime, and reduced repair costs. And as more and more functions are integrated into fewer components, the resulting system requires less power, produces less heat, and requires fewer mechanical connections—again resulting in greater system reliability.

LOWER COSTS

Using proven, reliable off-the-shelf components will reduce design costs, manufacturing costs, and time to market while increasing project viability and product reliability.

November 1993
Order Number: 241816-001

intel

INTRODUCTION

Intel microprocessors and peripherals provide a broad range of time-saving, energy-efficient, high-performance solutions to designers of both mobile and desktop microprocessor-based systems. Intel's microprocessor/peripheral interface delivers time and performance advantages to the designers of microprocessor-based systems, meeting their demand for greater performance, lower power consumption and a wider variety of built-in features for their customers.

IMPROVED SYSTEM PERFORMANCE

Intel offers an entire product line of microprocessors for desktop and mobile systems, ranging from the 25 MHz version of the Intel486™ SX processor to the high performance 66 MHz Intel486 DX2 processor. As Intel maintains its leadership in MIPS for microprocessor-based systems, it has managed to couple this superior performance with its sophisticated energy-efficient SL technology to meet the requirements of the EPA's Energy Star guidelines.

Intel has combined SL technology with desktop performance to enhance its entire Intel486 microprocessor family. Now, with built-in power management, SL Enhanced Intel486 CPU-based mobile systems achieve increases in battery life without compromising performance, while maintaining full compatibility with existing Intel architecture processors. SL Enhanced Intel486 CPUs for mobile and desktop systems are available in a full range of speeds (25 MHz to 66 MHz), packages (PGA, SQFP, TQFP), and voltage (5V, 3.3V) to meet any system design requirement.

REDUCED TIME TO MARKET

Intel's unrivaled motherboard for Intel486 SX, DX, and DX2 microprocessor-based desktop systems and scalable architecture for mobile systems reduces the development effort required to produce an entire product line of high-performance systems, thereby reducing time to market.

Intel offers a wide variety of off-the-shelf components to fulfill the requirements of system designers while simplifying the implementation of their designs. Off-the-shelf system solutions greatly decrease the potential risk of costly project delays due to component incompatibility. These system solutions greatly reduce the amount of time required to design, debug, manufacture and test microprocessor-based systems.

INCREASED RELIABILITY

At Intel, the rate of failure for devices is carefully tracked. Higher reliability is a tangible goal that translates to higher reliability for your product, reduced downtime, and reduced repair costs. And as more and more functions are integrated into fewer components, the resulting system requires less power, produces less heat, and requires fewer mechanical connections—again resulting in greater system reliability.

LOWER COSTS

Using proven, reliable off-the-shelf components will reduce design costs, manufacturing costs, and time to market while increasing project visibility and product reliability.

Intel486™ Microprocessor

2

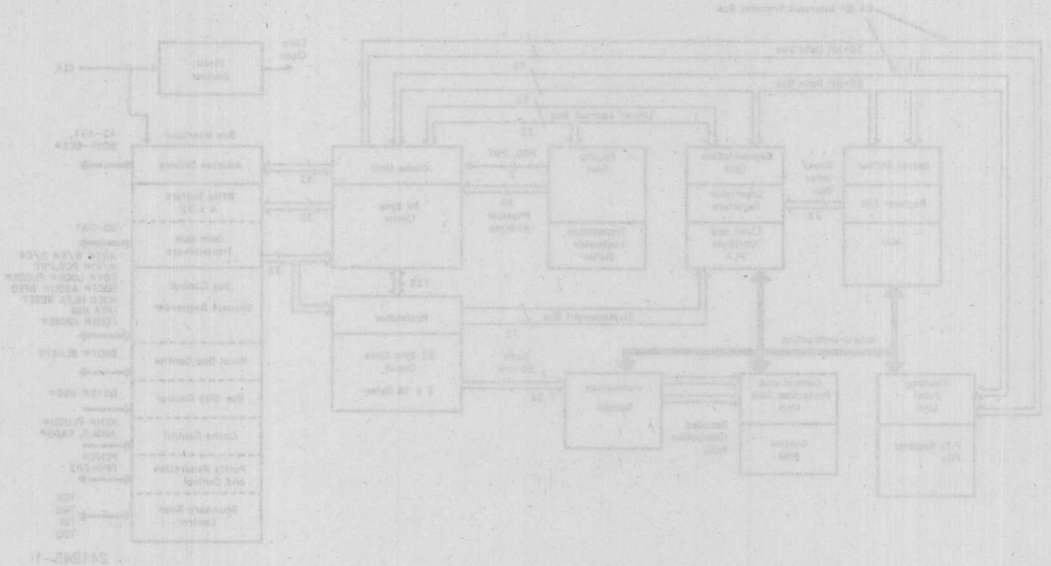
2 Intel486™ Microprocessor

INTRODUCTION TO THE Intel486™ MICROPROCESSOR FAMILY

In addition to the standard Intel486™ microprocessors (Intel486 SX, Intel486 DX and Intel486 DX2 CPUs), Intel is enhancing its entire Intel486 microprocessor family with the energy-efficient SL technology. The new SL Enhanced Intel486 microprocessor family enables built-in power management while maintaining full compatibility with existing Intel architecture processors.

The SL Enhanced Intel486 microprocessor packages are identified with the following laser mark: "&E". If the Intel486 SX, DX or DX2 microprocessor is lasermarked with "&E" on the package, refer to the SL Enhanced Intel486 Microprocessor Data Sheet Addendum on p. 2-715 for SL Enhanced Intel486 microprocessor specifications and power management features.

2



October 1993
Order Number: 241817-001

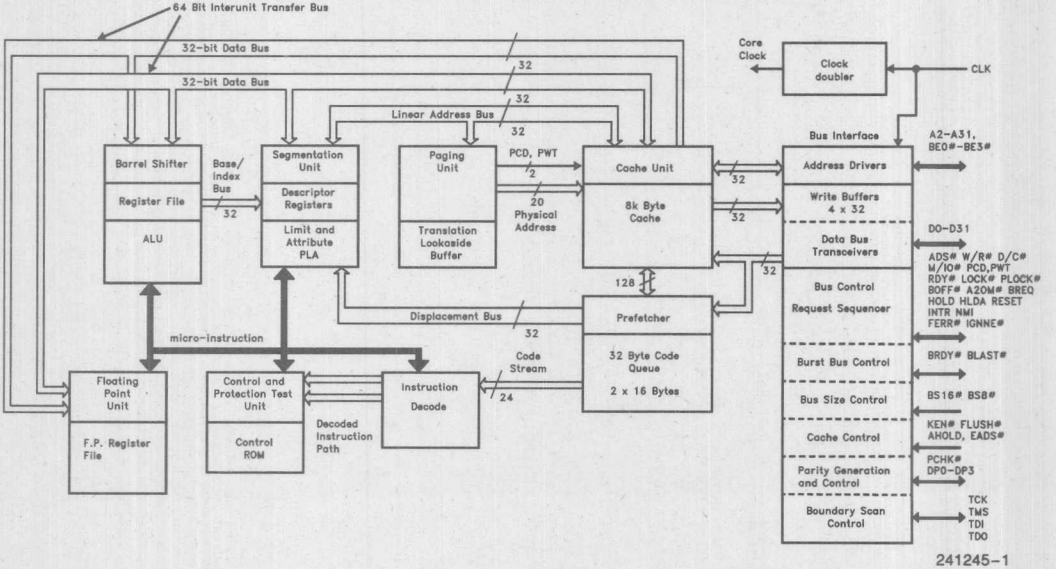
- **Binary Compatible with Large Software Base**
  - **— MS-DOS*, OS/2**, Windows**
  - **— UNIX*** System V/Intel386**
  - **— iRMX®, iRMK Kernels**
- **High Integration Enables On-Chip**
  - **— 8 Kbyte Code and Data Cache**
  - **— Floating Point Unit**
  - **— Paged, Virtual Memory Management**
- **Easy To Use**
  - **— Built-In Self Test**
  - **— Hardware Debugging Support**
- **168-Pin Grid Array Package**
  - **— Pin Compatible with Intel486™ DX Microprocessor**
- **IEEE 1149.1 Boundary Scan Compatibility**
- **High Performance Design**
  - **— 50 MHz/66 MHz Core Speed Using 25 MHz/33 MHz Bus Clocks**
  - **— RISC Integer Core with Frequent Instructions Executing in One Core Clock**
  - **— 80, 106 Mbyte/sec Burst Bus**
  - **— Dynamic Bus Sizing for 8-, 16-, and 32-Bit Busses**
  - **— Complete 32-Bit Architecture**
- **Multiprocessor Support**
  - **— Cache Consistency Protocols**
  - **— Support for Second Level Cache**

The Intel486 DX2 CPU offers the highest performance for DOS, OS/2, Windows, and UNIX System V/Intel386 applications. It is 100% binary compatible with the Intel386™ CPU. Over one million transistors integrate the RISC integer core, 8 Kbyte cache memory, floating point hardware, and memory management on-chip while retaining binary compatibility with previous members of the Intel386/Intel486 architectural family. The RISC integer core executes frequently-used instructions in one core clock cycle, providing leadership performance levels. An 8 Kbyte unified code and data cache allow the high performance levels to be sustained. A 106 MByte/sec burst bus at 33 MHz bus clock ensures high system throughput even with inexpensive DRAMs.

New features enhance multiprocessing systems; new instructions speed manipulation of memory-based semaphores; and on-chip hardware ensures cache consistency and provides hooks for multilevel caches.

The built-in self-test extensively tests on-chip logic, cache memory, and the on-chip paging translation cache. Debug features include breakpoint traps on code execution and data accesses.

Intel486™ DX2 Microprocessor Pipelined 32-Bit Microarchitecture



241245-1

November 1993
Order Number: 241245-004

2

Intel386, i386, Intel387, i387, Intel486, i486, Intel487, i487 are trademarks of Intel Corporation.
*MS-DOS is a registered trademark of Microsoft Corporation.
**OS/2 and Windows are trademarks of Microsoft Corporation.
***UNIX is a trademark of UNIX Systems Laboratories.

Quick Reference to Chapters

CONTENTS	PAGE
1.0 INTRODUCTION	2-12
2.0 ARCHITECTURAL OVERVIEW	2-19
3.0 REAL MODE ARCHITECTURE	2-50
4.0 PROTECTED MODE ARCHITECTURE	2-52
5.0 ON-CHIP CACHE	2-80
6.0 HARDWARE INTERFACE	2-85
7.0 BUS OPERATION	2-98
8.0 TESTABILITY	2-131
9.0 DEBUGGING SUPPORT	2-149

CONTENTS	PAGE
10.0 INSTRUCTION SET SUMMARY	2-154
11.0 DIFFERENCES WITH THE Intel386™ MICROPROCESSOR	2-181
12.0 Pentium™ OVERDRIVE™ PROCESSOR SOCKET	2-182
13.0 CONVERTING AN EXISTING Intel486™ CPU-BASED SYSTEM	2-192
14.0 ELECTRICAL DATA	2-197
15.0 MECHANICAL DATA	2-206
16.0 SUGGESTED SOURCES FOR Intel486™ ACCESSORIES	2-209
17.0 REVISION HISTORY	2-210

Intel486™ DX2 Microprocessor

CONTENTS

PAGE

1.0 INTRODUCTION 2-12
Pinout 2-10
Brief Pin Descriptions 2-13

2.0 ARCHITECTURAL OVERVIEW 2-19
2.1 Register Set 2-19
2.1.1 Base Architecture Registers 2-20
2.1.2 System Level Registers 2-24
2.1.3 Floating Point Registers 2-28
2.1.4 Debug and Test Registers 2-35
2.1.5 Register Accessibility 2-35
2.1.6 Compatibility 2-36
2.2 Instruction Set 2-37
2.3 Memory Organization 2-37
2.3.1 Address Spaces 2-37
2.3.2 Segment Register Usage 2-38
2.4 I/O Space 2-38
2.5 Addressing Modes 2-39
2.5.1 Addressing Modes Overview 2-39
2.5.2 Register and Immediate Modes 2-39
2.5.3 32-Bit Memory Addressing Modes 2-39
2.5.4 Differences between 16- and 32-Bit Addresses 2-41
2.6 Data Formats 2-41
2.6.1 Data Types 2-41
2.6.2 Little Endian vs Big Endian Data Formats 2-45
2.7 Interrupts 2-45
2.7.1 Interrupts and Exceptions 2-45
2.7.2 Interrupt Processing 2-45
2.7.3 Maskable Interrupt 2-46
2.7.4 Non-Maskable Interrupt 2-47
2.7.5 Software Interrupts 2-47
2.7.6 Interrupt and Exception Priorities 2-47
2.7.7 Instruction Restart 2-48

CONTENTS

PAGE

2.7.8 Double Fault 2-48
2.7.9 Floating Point Interrupt Vectors 2-48

3.0 REAL MODE ARCHITECTURE 2-50
3.1 Real Mode Introduction 2-50
3.2 Memory Addressing 2-51
3.3 Reserved Locations 2-51
3.4 Interrupts 2-51
3.5 Shutdown and Halt 2-51

4.0 PROTECTED MODE ARCHITECTURE 2-52
4.1 Introduction 2-52
4.2 Addressing Mechanism 2-52
4.3 Segmentation 2-53
4.3.1 Segmentation Introduction 2-53
4.3.2 Terminology 2-53
4.3.3 Descriptor Tables 2-53
4.3.4 Descriptors 2-54
4.4 Protection 2-63
4.4.1 Protection Concepts 2-63
4.4.2 Rules of Privilege 2-63
4.4.3 Privilege Levels 2-63
4.4.4 Privilege Level Transfers 2-66
4.4.5 Call Gates 2-67
4.4.6 Task Switching 2-67
4.4.7 Initialization and Transition to Protected Mode 2-68
4.4.8 Tools for Building Protected Systems 2-69
4.5 Paging 2-69
4.5.1 Paging Concepts 2-69
4.5.2 Paging Organization 2-70
4.5.3 Page Level Protection (R/W, U/S Bits) 2-71
4.5.4 Page Cacheability (PWT, PCD Bits) 2-72
4.5.5 Translation Lookaside Buffer 2-72
4.5.6 Paging Operation 2-73
4.5.7 Operating System Responsibilities 2-74

CONTENTS

PAGE

4.6 Virtual 8086 Environment 2-74
4.6.1 Executing 8086 Programs 2-74
4.6.2 Virtual 8086 Addressing Mechanism 2-74
4.6.3 Paging in Virtual Mode 2-74
4.6.4 Protection and Virtual 8086 Mode to I/O Permission Bitmap 2-75
4.6.5 Interrupt Handling 2-76
4.6.6 Entering and Leaving Virtual 8086 Mode 2-77

5.0 ON-CHIP CACHE 2-80
5.1 Cache Organization 2-80
5.2 Cache Control 2-81
5.3 Cache Line Fills 2-81
5.4 Cache Line Invalidations 2-82
5.5 Cache Replacement 2-82
5.6 Page Cacheability 2-83
5.7 Cache Flushing 2-84
5.8 Caching Translation Lookaside Buffer Entries 2-84

6.0 HARDWARE INTERFACE 2-85
6.1 Introduction 2-85
6.2 Signal Descriptions 2-86
6.2.1 Clock (CLK) 2-86
6.2.2 Address Bus (A31–A2, BE0#–BE3#) 2-86
6.2.3 Data Lines (D31–D0) 2-87
6.2.4 Parity 2-87
6.2.5 Bus Cycle Definition 2-87
6.2.6 Bus Control 2-88
6.2.7 Burst Control 2-88
6.2.8 Interrupt Signals 2-89
6.2.9 Bus Arbitration Signals 2-89
6.2.10 Cache Invalidation 2-90
6.2.11 Cache Control 2-91
6.2.12 Page Cacheability Outputs (PWT, PCD) 2-91
6.2.13 Numeric Error Reporting 2-91
6.2.14 Bus Size Control (BS16#, BS8#) 2-92

CONTENTS

PAGE

6.2.15 Address Bit 20 Mask (A20M#) 2-92
6.2.16 Boundary Scan Test Signals 2-92
6.3 Write Buffers 2-93
6.3.1 Write Buffers and I/O Cycles 2-94
6.3.2 Write Buffers Implications on Locked Bus Cycles 2-94
6.4 Interrupt and Non-Maskable Interrupt Interface 2-94
6.4.1 Interrupt Logic 2-94
6.4.2 NMI Logic 2-95
6.5 Reset and Initialization 2-95
6.5.1 Pin State during Reset 2-96

7.0 BUS OPERATION 2-98
7.1 Data Transfer Mechanism 2-98
7.1.1 Memory and I/O Spaces 2-98
7.1.2 Memory and I/O Space Organization 2-99
7.1.3 Dynamic Data Bus Sizing ... 2-100
7.1.4 Interfacing with 8-, 16- and 32-bit Memories 2-101
7.1.5 Dynamic Bus Sizing during Cache Line Fills 2-103
7.1.6 Operand Alignment 2-103
7.2 Bus Functional Description 2-104
7.2.1 Non-Cacheable Non-Burst Single Cycle 2-104
7.2.2 Multiple and Burst Cycle Bus Transfers 2-105
7.2.3 Cacheable Cycles 2-109
7.2.4 Burst Mode Details 2-112
7.2.5 8- and 16-Bit Cycles 2-116
7.2.6 Locked Cycles 2-118
7.2.7 Pseudo-Locked Cycles 2-119
7.2.8 Invalidate Cycles 2-119
7.2.9 Bus Hold 2-123
7.2.10 Interrupt Acknowledge 2-123
7.2.11 Special Bus Cycles 2-126
7.2.12 Bus Cycle Restart 2-127
7.2.13 Bus States 2-128
7.2.14 Floating Point Error Handling 2-129

CONTENTS

PAGE

8.0 TESTABILITY 2-131
8.1 Built-In Self Test (BIST) 2-131
8.2 On-Chip Cache Testing 2-131
8.2.1 Cache Testing Registers TR3, TR4 and TR5 2-131
8.2.2 Cache Testability Write 2-132
8.2.3 Cache Testability Read 2-134
8.2.4 Flush Cache 2-134
8.3 Translation Lookaside Buffer (TLB) Testing 2-134
8.3.1 Translation Lookaside Buffer Organization 2-134
8.3.2 TLB Test Registers: TR6 and TR7 2-135
8.3.3 TLB Write Test 2-137
8.3.4 TLB Lookup Test 2-137
8.4 3-state Output Test Mode 2-137
8.5 Intel486™ DX2 Microprocessor Boundary Scan (JTAG) 2-137
8.5.1 Boundary Scan Architecture 2-138
8.5.2 Data Registers 2-138
8.5.3 Instruction Register 2-139
8.5.4 Test Access Port (TAP) Controller 2-141
8.5.5 Boundary Scan Register Cell 2-144
8.5.6 TAP Controller Initialization 2-144
8.5.7 Boundary Scan Description Language (BSDL) 2-145

9.0 DEBUGGING SUPPORT 2-149
9.1 Breakpoint Instructions 2-149
9.2 Single Step Instructions 2-149
9.3 Debug Registers 2-149
9.3.1 Linear Address Breakpoint Registers 2-150
9.3.2 Debug Control Register 2-150
9.3.3 Debug Status Register 2-153
9.3.4 Use of Resume Flag (RF) in Flag Register 2-153

CONTENTS

PAGE

10.0 INSTRUCTION SET SUMMARY .. 2-154
10.1 Intel486™ DX2 Microprocessor Instruction Encoding and Clock Count Summary 2-154
10.2 Instruction Encoding 2-173
10.2.1 Overview 2-173
10.2.2 32-Bit Extensions of the Instruction Set 2-174
10.2.3 Encoding of Integer Instruction Fields 2-174
10.2.4 Encoding of Floating Point Instruction Fields 2-180

11.0 DIFFERENCES WITH THE Intel386™ MICROPROCESSOR 2-181

12.0 Pentium™ OVERDRIVE™ PROCESSOR SOCKET 2-182
12.0.1 Pentium OverDrive Processor Overview 2-183
12.1 Pentium OverDrive Processor Circuit Design 2-183
12.2 Socket Layout 2-184
12.2.1 Backward Compatibility 2-184
12.2.2 Physical Dimensions 2-184
12.2.3 "End User Easy" 2-185
12.2.4 ZIF Socket Vendors 2-186
12.3 Thermal Design Considerations 2-186
12.4 BIOS and Software 2-187
12.4.1 Intel486™ DX2 Upgrade Processor Detection 2-187
12.4.2 Timing Dependent Loops .. 2-187
12.5 Test Requirements 2-187
12.6 Pentium OverDrive Processor Socket Pinout 2-187
12.6.1 Pinout 2-188
12.6.2 Pin Description 2-190
12.6.3 Reserved Pin Specification 2-190
12.7 D.C./A.C. Specifications 2-191

CONTENTS

PAGE

13.0 CONVERTING AN EXISTING Intel486™ CPU-BASED SYSTEM ... 2-192

14.0 ELECTRICAL DATA 2-197

14.1 Power and Grounding 2-197

14.2 Maximum Ratings 2-197

14.3 D.C. Specifications 2-198

14.4 A.C. Specifications 2-198

CONTENTS

PAGE

15.0 MECHANICAL DATA 2-206

15.1 Package Thermal Specifications 2-207

16.0 SUGGESTED SOURCES FOR Intel486™ DX2 ACCESSORIES 2-209

17.0 REVISION HISTORY 2-210

DATA SHEET DESIGNATONS

Intel uses various data sheet markings to designate each phase of the document as it relates to the product. The marking appears in the upper, right-hand corner of the data sheet. The following is the definition of these markings:

Data Sheet Marking	Description
Product Preview	Contains information on products in the design phase of development. Do not finalize a design with this information. Revised information will be published when the product becomes available.
Advanced Information	Contains information on products being sampled or in the initial production phase of development.*
Preliminary	Contains preliminary information on new products in production.*
No Marking	Contains information on products in full production.*

*Specifications within these data sheets are subject to change without notice. Verify with your local Intel sales office that you have the latest data sheet before finalizing a design.

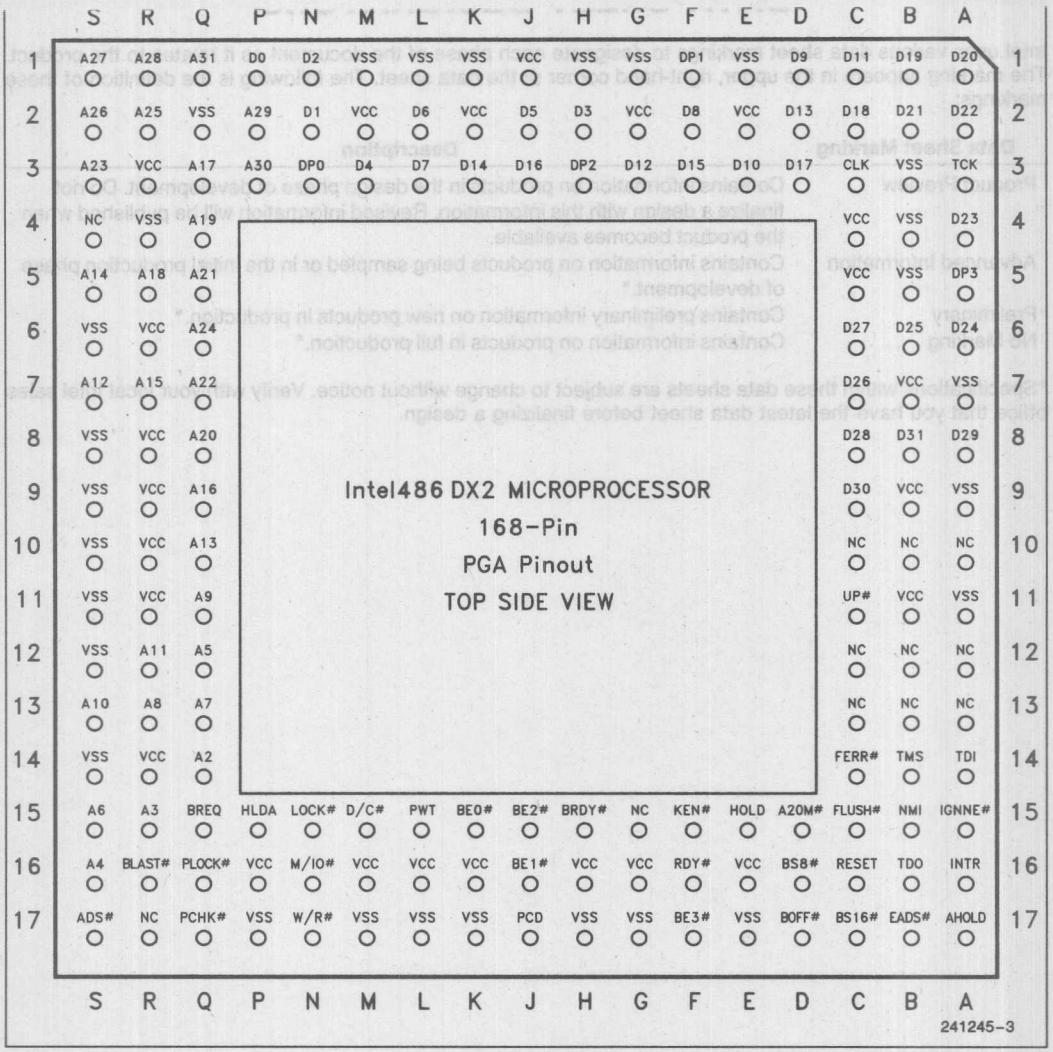

S R Q P N M L K J H G F E D C B A
1 A27 A28 A31 D0 D2 VSS VSS VSS VCC VSS VSS DP1 VSS D9 D11 D19 D20 1
2 A26 A25 VSS A29 D1 VCC D6 VCC D5 D3 VCC D8 VCC D13 D18 D21 D22 2
3 A23 VCC A17 A30 DP0 D4 D7 D14 D16 DP2 D12 D15 D10 D17 CLK VSS TCK 3
4 NC VSS A19 VCC VSS D23 4
5 A14 A18 A21 VCC VSS DP3 5
6 VSS VCC A24 D27 D25 D24 6
7 A12 A15 A22 D26 VCC VSS 7
8 VSS VCC A20 D28 D31 D29 8
9 VSS VCC A16 D30 VCC VSS 9
10 VSS VCC A13 NC NC NC 10
11 VSS VCC A9 UP# VCC VSS 11
12 VSS A11 A5 NC NC NC 12
13 A10 A8 A7 NC NC NC 13
14 VSS VCC A2 FERR# TMS TDI 14
15 A6 A3 BREQ HLDA LOCK# D/C# PWT BE0# BE2# BRDY# NC KEN# HOLD A20M# FLUSH# NMI IGNNE# 15
16 A4 BLAST# PLOCK# VCC M/IO# VCC VCC VCC BE1# VCC VCC RDY# VCC BS8# RESET TDO INTR 16
17 ADS# NC PCHK# VSS W/R# VSS VSS VSS PCD VSS VSS BE3# VSS BOFF# BS16# EADS# AHOLD 17
S R Q P N M L K J H G F E D C B A
Intel486 DX2 MICROPROCESSOR
168-Pin
PGA Pinout
TOP SIDE VIEW
241245-3

Figure 1.1

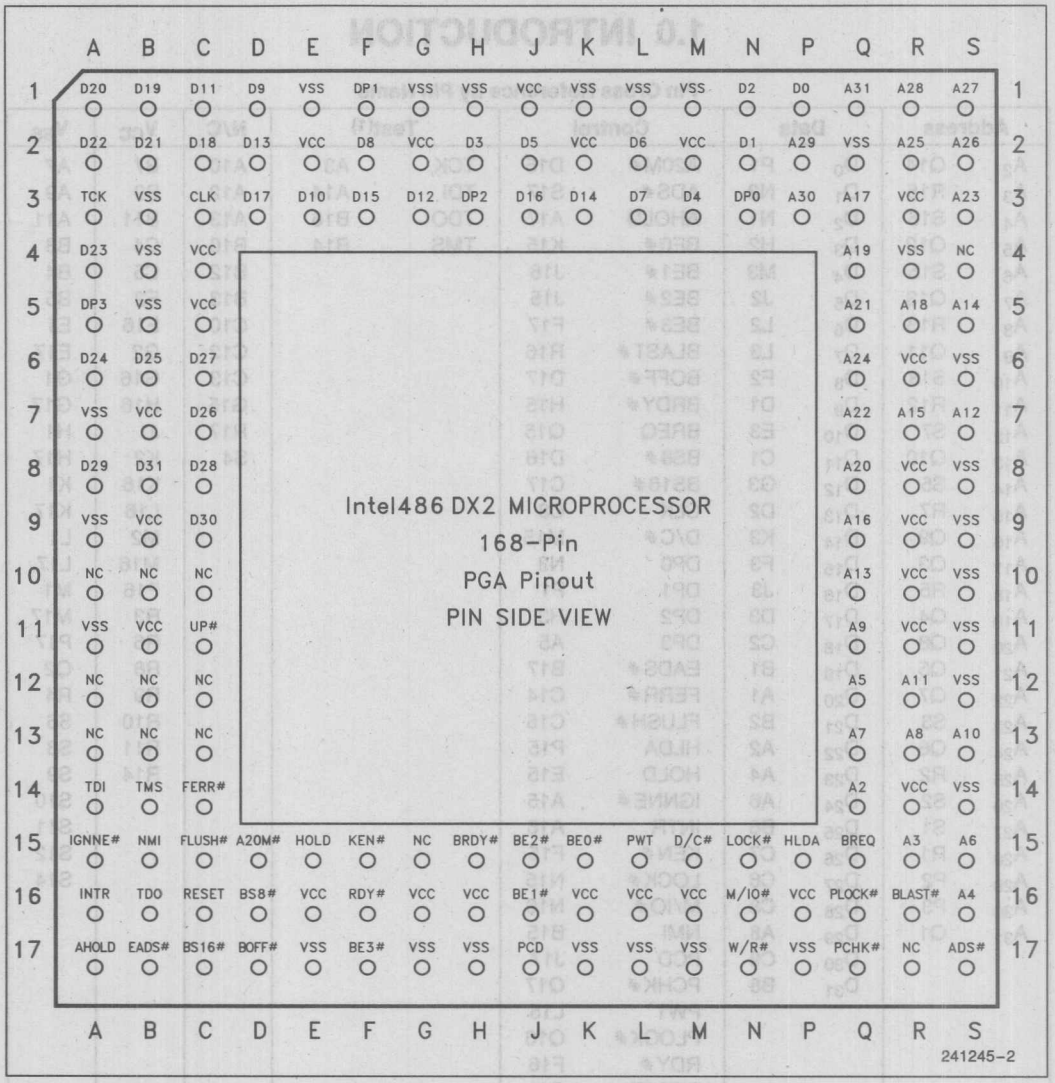

A B C D E F G H J K L M N P Q R S
1 D20 D19 D11 D9 VSS DP1 VSS VSS VCC VSS VSS VSS D2 D0 A31 A28 A27
2 D22 D21 D18 D13 VCC D8 VCC D3 D5 VCC D6 VCC D1 A29 VSS A25 A26
3 TCK VSS CLK D17 D10 D15 D12 DP2 D16 D14 D7 D4 DP0 A30 A17 VCC A23
4 D23 VSS VCC A19 VSS NC
5 DP3 VSS VCC A21 A18 A14
6 D24 D25 D27 A24 VCC VSS
7 VSS VCC D26 A22 A15 A12
8 D29 D31 D28 A20 VCC VSS
9 VSS VCC D30 A16 VCC VSS
10 NC NC NC A13 VCC VSS
11 VSS VCC UP# A9 VCC VSS
12 NC NC NC A5 A11 VSS
13 NC NC NC A7 A8 A10
14 TDI TMS FERR# A2 VCC VSS
15 IGNNE# NMI FLUSH# A20M# HOLD KEN# NC BRDY# BE2# BE0# PWT D/C# LOCK# HLDA BREQ A3 A6
16 INTR TDO RESET BS8# VCC RDY# VCC VCC BE1# VCC VCC VCC M/IO# VCC PLOCK# BLAST# A4
17 AHOLD EADS# BS16# BOFF# VSS BE3# VSS VSS PCD VSS VSS VSS W/R# VSS PCHK# NC ADS#
Intel486 DX2 MICROPROCESSOR
168-Pin
PGA Pinout
PIN SIDE VIEW
241245-2

Figure 1.2

1.0 INTRODUCTION

Pin Cross Reference by Pin Name

Address		Data		Control		Test(1)		N/C	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	TCK	A3	A10	B7	A7
A_3	R15	D_1	N2	ADS#	S17	TDI	A14	A12	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	TDO	B16	A13	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	TMS	B14	B10	C4	B3
A_6	S15	D_4	M3	BE1#	J16			B12	C5	B4
A_7	Q13	D_5	J2	BE2#	J15			B13	E2	B5
A_8	R13	D_6	L2	BE3#	F17			C10	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16			C12	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17			C13	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15			G15	H16	G17
A_{12}	S7	D_{10}	E3	BREQ	Q15			R17	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16			S4	K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17				K16	K1
A_{15}	R7	D_{13}	D2	CLK	C3				L16	K17
A_{16}	Q9	D_{14}	K3	D/C#	M15				M2	L1
A_{17}	Q3	D_{15}	F3	DP0	N3				M16	L17
A_{18}	R5	D_{16}	J3	DP1	F1				P16	M1
A_{19}	Q4	D_{17}	D3	DP2	H3				R3	M17
A_{20}	Q8	D_{18}	C2	DP3	A5				R6	P17
A_{21}	Q5	D_{19}	B1	EADS#	B17				R8	Q2
A_{22}	Q7	D_{20}	A1	FERR#	C14				R9	R4
A_{23}	S3	D_{21}	B2	FLUSH#	C15				R10	S6
A_{24}	Q6	D_{22}	A2	HLDA	P15				R11	S8
A_{25}	R2	D_{23}	A4	HOLD	E15				R14	S9
A_{26}	S2	D_{24}	A6	IGNNE#	A15					S10
A_{27}	S1	D_{25}	B6	INTR	A16					S11
A_{28}	R1	D_{26}	C7	KEN#	F15					S12
A_{29}	P2	D_{27}	C6	LOCK#	N15					S14
A_{30}	P3	D_{28}	C8	M/IO#	N16					
A_{31}	Q1	D_{29}	A8	NMI	B15					
		D_{30}	C9	PCD	J17					
		D_{31}	B8	PCHK#	Q17					
				PWT	L15					
				PLOCK#	Q16					
				RDY#	F16					
				RESET	C16					
				UP#(1)	C11					
				W/R#	N17					

NOTE:
1. These pins were No-Connects on the 25 MHz and 33 MHz Intel486 DX microprocessors. For compatibility with old designs they can still be left unconnected.

QUICK PIN REFERENCE

What follows is a brief pin description. For detailed signal descriptions refer to Section 6.

Symbol	Type	Name and Function
CLK	I	*Clock* provides the fundamental timing for the bus interface unit and is multiplied by two (2x) to provide the internal frequency for the Intel486 DX2. All external timing parameters are specified with respect to the rising edge of CLK.
ADDRESS BUS		
A31–A4 A2–A3	I/O O	A31–A2 are the *address lines* of the microprocessor. A31–A2, together with the byte enables BE0#–BE3#, define the physical area of memory or input/output space accessed. Address lines A31–A4 are used to drive addresses into the microprocessor to perform cache line invalidations. Input signals must meet setup and hold times t_{22} and t_{23}. A31–A2 are not driven during bus or address hold.
BE0–3#	O	The *byte enable* signals indicate active bytes during read and write cycles. During the first cycle of a cache fill, the external system should assume that all byte enables are active. BE3# applies to D24–D31, BE2# applies to D16–D23, BE1# applies to D8–D15 and BE0# applies to D0–D7. BE0#–BE3# are active LOW and are not driven during bus hold.
DATA BUS		
D31–D0	I/O	These are the *data lines* for the Intel486 DX2 microprocessor. Lines D0–D7 define the least significant byte of the data bus while lines D24–D31 define the most significant byte of the data bus. These signals must meet setup and hold times t_{22} and t_{23} for proper operation on reads. These pins are driven during the second and subsequent clocks of write cycles.
DATA PARITY		
DP0–DP3	I/O	There is one *data parity* pin for each byte of the data bus. Data parity is generated on all write data cycles with the same timing as the data driven by the Intel486 DX2 microprocessor. Even parity information must be driven back into the microprocessor on the data parity pins with the same timing as read information to insure that the correct parity check status is indicated by the Intel486 DX2 microprocessor. The signals read on these pins do not affect program execution. Input signals must meet setup and hold times t_{22} and t_{23}. DP0–DP3 should be connected to V_{CC} through a pullup resistor in systems which do not use parity. DP0–DP3 are active HIGH and are driven during the second and subsequent clocks of write cycles.
PCHK#	O	*Parity Status* is driven on the PCHK# pin the clock after ready for read operations. The parity status is for data sampled at the end of the previous clock. A parity error is indicated by PCHK# being LOW. Parity status is only checked for enabled bytes as indicated by the byte enable and bus size signals. PCHK# is valid only in the clock immediately after read data is returned to the microprocessor. At all other times PCHK# is inactive (HIGH). PCHK# is never floated.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
BUS CYCLE DEFINITION		
M/IO# D/C# W/R#	O O O	The *memory/input-output, data/control* and *write/read* lines are the primary bus definition signals. These signals are driven valid as the ADS# signal is asserted.
		M/IO# D/C# W/R# Bus Cycle Initiated 0 0 0 Interrupt Acknowledge 0 0 1 Halt/Special Cycle 0 1 0 I/O Read 0 1 1 I/O Write 1 0 0 Code Read 1 0 1 Reserved 1 1 0 Memory Read 1 1 1 Memory Write
		The bus definition signals are not driven during bus hold and follow the timing of the address bus. Refer to Section 7.2.11 for a description of the special bus cycles.
LOCK#	O	The *bus lock* pin indicates that the current bus cycle is locked. The Intel486 DX2 microprocessor will not allow a bus hold when LOCK# is asserted (but address holds are allowed). LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after the last clock of the last locked bus cycle. The last locked cycle ends when ready is returned. LOCK# is active LOW and is not driven during bus hold. Locked read cycles will not be transformed into cache fill cycles if KEN# is returned active.
PLOCK#	O	The *pseudo-lock* pin indicates that the current bus transaction requires more than one bus cycle to complete. Examples of such operations are floating point long reads and writes (64 bits), segment table descriptor reads (64 bits), in addition to cache line fills (128 bits). The Intel486 DX2 microprocessor will drive PLOCK# active until the addresses for the last bus cycle of the transaction have been driven regardless of whether RDY# or BRDY# have been returned. Normally PLOCK# and BLAST# are inverse of each other. However during the first bus cycle of a 64-bit floating point write, both PLOCK# and BLAST# will be asserted. PLOCK# is a function of the BS8#, BS16# and KEN# inputs. PLOCK# should be sampled only in the clock ready is returned. PLOCK# is active LOW and is not driven during bus hold.
BUS CONTROL		
ADS#	O	The *address status* output indicates that a valid bus cycle definition and address are available on the cycle definition lines and address bus. ADS# is driven active in the same clock as the addresses are driven. ADS# is active LOW and is not driven during bus hold.
RDY#	I	The *non-burst ready* input indicates that the current bus cycle is complete. RDY# indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted data from the Intel486 DX2 microprocessor in response to a write. RDY# is ignored when the bus is idle and at the end of the first clock of the bus cycle. RDY# is active during address hold. Data can be returned to the processor while AHOLD is active. RDY# is active LOW, and is not provided with an internal pullup resistor. RDY# must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
BURST CONTROL		
BRDY#	I	The *burst ready input* performs the same function during a burst cycle that RDY# performs during a non-burst cycle. BRDY# indicates that the external system has presented valid data in response to a read or that the external system has accepted data in response to a write. BRDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle. BRDY# is sampled in the second and subsequent clocks of a burst cycle. The data presented on the data bus will be strobed into the microprocessor when BRDY# is sampled active. If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is prematurely interrupted BRDY# is active LOW and is provided with a small pullup resistor. BRDY# must satisfy the setup and hold times t_{16} and t_{17}.
BLAST#	O	The *burst last* signal indicates that the next time BRDY# is returned the burst bus cycle is complete. BLAST# is active for both burst and non-burst bus cycles. BLAST# is active LOW and is not driven during bus hold.
INTERRUPTS		
RESET	I	The *reset* input forces the Intel486 DX2 microprocessor to begin execution at a known state. The microprocessor cannot begin execution of instructions until at least 1 ms after V_{CC} and CLK have reached their proper DC and AC specifications. The RESET pin should remain active during this time to insure proper microprocessor operation. RESET is active HIGH. RESET is asynchronous but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
INTR	I	The *maskable interrupt* indicates that an external interrupt has been generated. If the internal interrupt flag is set in EFLAGS, active interrupt processing will be initiated. The Intel486 DX2 microprocessor will generate two locked interrupt acknowledge bus cycles in response to the INTR pin going active. INTR must remain active until the interrupt acknowledges have been performed to assure that the interrupt is recognized. INTR is active HIGH and is not provided with an internal pulldown resistor. INTR is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
NMI	I	The *non-maskable interrupt* request signal indicates that an external non-maskable interrupt has been generated. NMI is rising edge sensitive. NMI must be held LOW for at least four CLK periods before this rising edge. NMI is not provided with an internal pulldown resistor. NMI is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
BUS ARBITRATION		
BREQ	O	The *internal cycle pending* signal indicates that the Intel486 DX2 microprocessor has internally generated a bus request. BREQ is generated whether or not the Intel486 DX2 microprocessor is driving the bus. BREQ is active HIGH and is never floated.
HOLD	I	The *bus hold request* allows another bus master complete control of the Intel486 DX2 microprocessor bus. In response to HOLD going active the Intel486 DX2 microprocessor will float most of its output and input/output pins. HLDA will be asserted after completing the current bus cycle, burst cycle or sequence of locked cycles. The Intel486 DX2 microprocessor will remain in this state until HOLD is deasserted. HOLD is active high and is not provided with an internal pulldown resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper operation.
HLDA	O	*Hold acknowledge* goes active in response to a hold request presented on the HOLD pin. HLDA indicates that the Intel486 DX2 microprocessor has given the bus to another local bus master. HLDA is driven active in the same clock that the Intel486 DX2 microprocessor floats its bus. HLDA is driven inactive when leaving bus hold. HLDA is active HIGH and remains driven during bus hold.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
BUS ARBITRATION (Continued)		
BOFF#	I	The *backoff* input forces the Intel486 DX2 microprocessor to float its bus in the next clock. The microprocessor will float all pins normally floated during bus hold but HLDA will not be asserted in response to BOFF#. BOFF# has higher priority than RDY# or BRDY#; if both are returned in the same clock, BOFF# takes effect. The microprocessor remains in bus hold until BOFF# is negated. If a bus cycle was in progress when BOFF# was asserted the cycle will be restarted. BOFF# is active LOW and must meet setup and hold times t_{18} and t_{19} for proper operation.
CACHE INVALIDATION		
AHOLD	I	The *address hold* request allows another bus master access to the Intel486 DX2 microprocessor's address bus for a cache invalidation cycle. The Intel486 DX2 microprocessor will stop driving its address bus in the clock following AHOLD going active. Only the address bus will be floated during address hold, the remainder of the bus will remain active. AHOLD is active HIGH and is provided with a small internal pulldown resistor. For proper operation AHOLD must meet setup and hold times t_{18} and t_{19}.
EADS#	I	This signal indicates that a *valid external address* has been driven onto the Intel486 DX2 microprocessor address pins. This address will be used to perform an internal cache invalidation cycle. EADS# is active LOW and is provided with an internal pullup resistor. EADS# must satisfy setup and hold times t_{12} and t_{13} for proper operation.
CACHE CONTROL		
KEN#	I	The *cache enable* pin is used to determine whether the current cycle is cacheable. When the Intel486 DX2 microprocessor generates a cycle that can be cached and KEN# is active, the cycle will become a cache line fill cycle. Returning KEN# active one clock before ready during the last read in the cache line fill will cause the line to be placed in the on-chip cache. KEN# is active LOW and is provided with a small internal pullup resistor. KEN# must satisfy setup and hold times t_{14} and t_{15} for proper operation.
FLUSH#	I	The *cache flush* input forces the Intel486 DX2 microprocessor to flush its entire internal cache. FLUSH# is active low and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition in any specific clock. FLUSH# being sampled low in the clock before the falling edge of RESET causes the Intel486 DX2 microprocessor to enter the tri-state test mode.
PAGE CACHEABILITY		
PWT PCD	O O	The *page write-through* and *page cache disable* pins reflect the state of the page attribute bits, PWT and PCD, in the page table entry or page directory entry. If paging is disabled or for cycles that are not paged, PWT and PCD reflect the state of the PWT and PCD bits in control register 3. PWT and PCD have the same timing as the cycle definition pins (M/IO#, D/C# and W/R#). PWT and PCD are active HIGH and are not driven during bus hold. PCD is masked by the cache disable bit (CD) in Control Register 0.
NUMERIC ERROR REPORTING		
FERR#	O	The *floating point error* pin is driven active when a floating point error occurs. FERR# is similar to the ERROR# pin on the Intel387™ math coprocessor. FERR# is included for compatibility with systems using DOS type floating point error reporting. FERR# will not go active if FP errors are masked in FPU register. FERR# is active LOW, and is not floated during bus hold.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
NUMERIC ERROR REPORTING (Continued)		
IGNNE#	I	When the *ignore numeric error* pin is asserted the Intel486 DX2 microprocessor will ignore a numeric error and continue executing non-control floating point instructions, but FERR# will still be activated by the Intel486 DX2. When IGNNE# is deasserted the Intel486 DX2 microprocessor will freeze on a non-control floating point instruction, if a previous floating point instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set. IGNNE# is active LOW and is provided with a small internal pullup resistor. IGNNE# is asynchronous but setup and hold times t_{20} and t_{21} must be met to insure recognition on any specific clock.
BUS SIZE CONTROL		
BS16# BS8#	I I	The *bus size 16* and *bus size 8* pins (bus sizing pins) cause the Intel486 DX2 microprocessor to run multiple bus cycles to complete a request from devices that cannot provide or accept 32 bits of data in a single cycle. The bus sizing pins are sampled every clock. The state of these pins in the clock before ready is used by the Intel486 DX2 microprocessor to determine the bus size. These signals are active LOW and are provided with internal pullup resistors. These inputs must satisfy setup and hold times t_{14} and t_{15} for proper operation.
ADDRESS MASK		
A20M#	I	When the *address bit 20 mask* pin is asserted, the Intel486 DX2 microprocessor masks physical address bit 20 (A20) before performing a lookup to the internal cache or driving a memory cycle on the bus. A20M# emulates the address wraparound at one Mbyte which occurs on the 8086. A20M# is active LOW and should be asserted only when the processor is in real mode. This pin is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For proper operation, A20M# should be sampled high at the falling edge of RESET.
TEST ACCESS PORT		
TCK	I	*Test Clock* is an input to the Intel486 DX2 CPU and provides the clocking function required by the JTAG boundary scan feature. TCK is used to clock state information and data into and out of the component. State select information and data are clocked into the component on the rising edge of TCK on TMS and TDI, respectively. Data is clocked out of the part on the falling edge of TCK on TDO.
TDI	I	*Test Data Input* is the serial input used to shift JTAG instructions and data into the component. TDI is sampled on the rising edge of TCK, during the SHIFT-IR and the SHIFT-DR TAP controller states. During all other tap controller states, TDI is a "don't care".
TDO	O	*Test Data Output* is the serial output used to shift JTAG instructions and data out of the component. TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT-DR TAP controller states. At all other times TDO is driven to the high impedance state.
TMS	I	*Test Mode Select* is decoded by the JTAG TAP (Tap Access Port) to select the operation of the test logic. TMS is sampled on the rising edge of TCK. To guarantee deterministic behavior of the TAP controller TMS is provided with an internal pull-up resistor.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
POWER DOWN MODE (UPGRADE PROCESSOR SUPPORT)		
UP#	I	The *Upgrade Present* pin forces the Intel486 DX2 to 3-state all of its outputs and enter the power down mode. When the Upgrade Present pin is sampled asserted by the CPU in the clock before the falling edge of RESET, the power down mode is enabled. UP# has no effect on the power down status except during this edge. The CPU is also forced to 3-state all of it's outputs immediately in response to this signal. The UP# signal must remain asserted in order to keep the pins 3-stated. UP# is active low and is provided with an internal pull-up resistor.

Table 1.1. Output Pins

Name	Active Level	When Floated
BREQ	HIGH	
HLDA	HIGH	
BE0#–BE3#	LOW	Bus Hold
PWT, PCD	HIGH	Bus Hold
W/R#, D/C#, M/IO#	HIGH	Bus Hold
LOCK#	LOW	Bus Hold
PLOCK#	LOW	Bus Hold
ADS#	LOW	Bus Hold
BLAST#	LOW	Bus Hold
PCHK#	LOW	
FERR#	LOW	
RES_B	LOW	
A2–A3	HIGH	Bus, Address Hold

Table 1.2. Input Pins

Name	Active Level	Synchronous/ Asynchronous
CLK		
RESET	HIGH	Asynchronous
HOLD	HIGH	Synchronous
AHOLD	HIGH	Synchronous
EADS#	LOW	Synchronous
BOFF#	LOW	Synchronous
FLUSH#	LOW	Asynchronous
A20M#	LOW	Asynchronous
BS16#, BS8#	LOW	Synchronous
KEN#	LOW	Synchronous
RDY#	LOW	Synchronous
BRDY#	LOW	Synchronous
INTR	HIGH	Asynchronous
NMI	HIGH	Asynchronous
IGNNE#	LOW	Asynchronous
RES_A	LOW	Asynchronous
UP#	LOW	Asynchronous

Table 1.3. Input/Output Pins

Name	Active Level	When Floated
D0–D31	HIGH	Bus Hold
DP0–DP3	HIGH	Bus Hold
A4–A31	HIGH	Bus, Address Hold

Table 1.4. Test Pins

Name	Input or Output	Sampled/ Driven On
TCK	Input	N/A
TDI	Input	Rising Edge of TCK
TDO	Output	Falling Edge of TCK
TMS	Input	Rising Edge of TCK

2.0 ARCHITECTURAL OVERVIEW

The Intel486 DX2 microprocessor is a fully compatible member of the Intel486 family.

The Intel486 microprocessor family is a 32-bit architecture with on-chip memory management, floating point and cache memory units.

The Intel486 DX microprocessor contains all the features of the Intel386™ microprocessor with enhancements to increase performance. The instruction set includes the complete Intel386 microprocessor instruction set along with extensions to serve new applications. The on-chip memory management unit (MMU) is completely compatible with the Intel386 microprocessor MMU. The Intel486 DX microprocessor brings the Intel387™ math coprocessor on-chip. All software written for the Intel386 microprocessor, Intel387 math coprocessor and previous members of the 86/87 architectural family will run on the Intel486 DX microprocessor without any modifications.

Several enhancements have been added to the Intel486 DX microprocessor to increase performance. On-chip cache memory allows frequently used data and code to be stored on-chip reducing accesses to the external bus. A clock doubler has been added to speed up internal operations to twice that of an Intel486 DX microprocessor running with the same bus clock. RISC design techniques have been used to reduce instruction cycle times. A burst bus feature enables fast cache fills. All of these features, combined, lead to performance greater than triple that of a Intel386 microprocessor.

The memory management unit (MMU) consists of a segmentation unit and a paging unit. Segmentation allows management of the logical address space by providing easy data and code relocatibility and efficient sharing of global resources. The paging mechanism operates beneath segmentation and is transparent to the segmentation process. Paging is optional and can be disabled by system software. Each segment can be divided into one or more 4 Kbyte segments. To implement a virtual memory system, the Intel486 DX microprocessor supports full restartability for all page and segment faults.

Memory is organized into one or more variable length segments, each up to four gigabytes (2^{32} bytes) in size. A segment can have attributes associated with it which include its location, size, type (i.e., stack, code or data), and protection characteristics. Each task on a Intel486 DX microprocessor can have a maximum of 16,381 segments, each up to four gigabytes in size. Thus each task has a maximum of 64 terabytes (trillion bytes) of virtual memory.

The segmentation unit provides four-levels of protection for isolating and protecting applications and the operating system from each other. The hardware enforced protection allows the design of systems with a high degree of integrity.

The Intel486 DX microprocessor has two modes of operation: Real Address Mode (Real Mode) and Protected Mode Virtual Address Mode (Protected Mode). In Real Mode the Intel486 DX microprocessor operates as a very fast 8086. Real Mode is required primarily to set up the processor for Protected Mode operation. Protected Mode provides access to the sophisticated memory management paging and privilege capabilities of the processor.

Within Protected Mode, software can perform a task switch to enter into tasks designated as Virtual 8086 Mode tasks. Each virtual 8086 task behaves with 8086 semantics, allowing 8086 software (an application program or an entire operating system) to execute.

The on-chip floating point unit operates in parallel with the arithmetic and logic unit and provides arithmetic instructions for a variety of numeric data types. It executes numerous built-in transcendental functions (e.g., tangent, sine, cosine, and log functions). The floating point unit fully conforms to the ANSI/IEEE standard 754-1985 for floating point arithmetic.

The on-chip cache is 8 Kbytes in size. It is 4-way set associative and follows a write-through policy. The on-chip cache includes features to provide flexibility in external memory system design. Individual pages can be designated as cacheable or non-cacheable by software or hardware. The cache can also be enabled and disabled by software or hardware.

Finally the Intel486 DX2 microprocessor has features to facilitate high performance hardware designs. The 1x bus clock eases high frequency board level designs. While the 2x clock doubler improves execution performance without increasing the board design complexity. This 2x clock doubler enhances all operations operating out of the cache and/or not blocked by external bus accesses. The burst bus feature enables fast cache fills. These features are described beginning in Section 6.

2.1 Register Set

The Intel486 DX microprocessor register set includes all the registers contained in the Intel386 microprocessor and the Intel387 math coprocessor.

The register set can be split into the following categories:

- Base Architecture Registers
  - General Purpose Registers
  - Instruction Pointer
  - Flags Register
  - Segment Registers
- Systems Level Registers
  - Control Registers
  - System Address Registers
- Floating Point Registers
  - Data Registers
  - Tag Word
  - Status Word
  - Instruction and Data Pointers
  - Control Word
- Debug and Test Registers

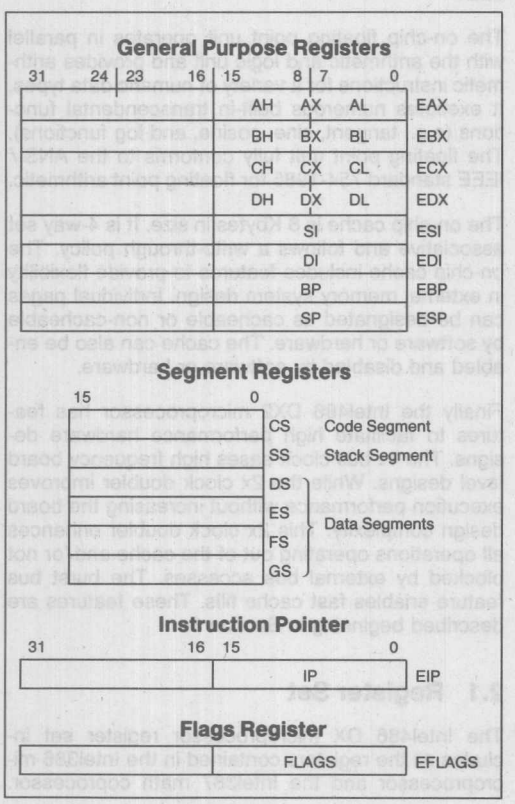


Figure 2.1. Base Architecture Registers

The base architecture and floating point registers are accessible by the applications program. The system level registers are only accessible at privilege level 0 and are used by the systems level program. The debug and test registers are also only accessible at privilege level 0.

2.1.1 BASE ARCHITECTURE REGISTERS

Figure 2.1 shows the Intel486 DX2 microprocessor base architecture registers. The contents of these registers are task-specific and are automatically loaded with a new context upon a task switch operation.

The base architecture includes six directly accessible descriptors, each specifying a segment up to 4 Gbytes in size. The descriptors are indicated by the selector values placed in the Intel486 DX2 microprocessor segment registers. Various selector values can be loaded as a program executes.

The selectors are also task-specific, so the segment registers are automatically loaded with new context upon a task switch operation.

2.1.1.1 General Purpose Registers

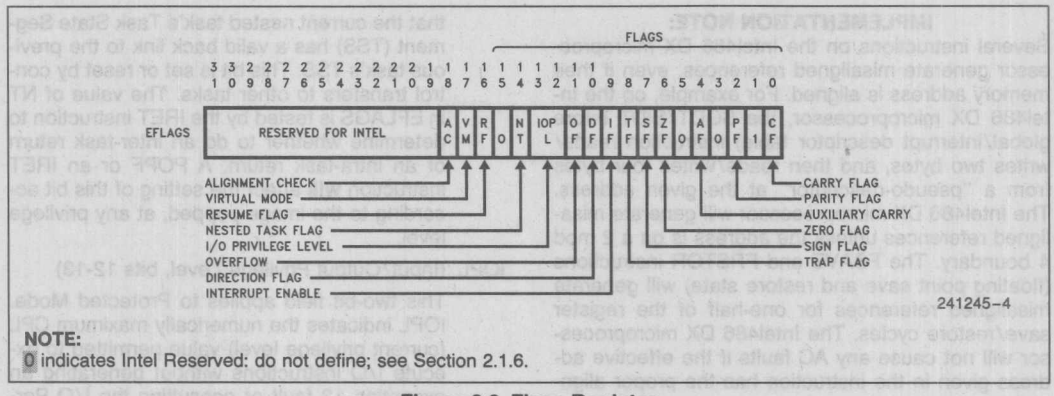
The eight 32-bit general purpose registers are shown in Figure 2.1. These registers hold data or address quantities. The general purpose registers can support data operands of 1, 8, 16 and 32 bits, and bit fields of 1 to 32 bits. Address operands of 16 and 32 bits are supported. The 32-bit registers are named EAX, EBX, ECX, EDX, ESI, EDI, EBP and ESP.

The least significant 16 bits of the general purpose registers can be accessed separately by using the 16-bit names of the registers AX, BX, CX, DX, SI, DI, BP and SP. The upper 16 bits of the register are not changed when the lower 16 bits are accessed separately.

Finally 8-bit operations can individually access the lowest byte (bits 0–7) and the higher byte (bits 8–15) of the general purpose registers AX, BX, CX and DX. The lowest bytes are named AL, BL, CL and DL respectively. The higher bytes are named AH, BH, CH and DH respectively. The individual byte accessibility offers additional flexibility for data operations but is not used for effective address calculation.

2.1.1.2 Instruction Pointer

The instruction pointer, shown in Figure 2.1, is a 32-bit register named EIP. EIP holds the offset of the next instruction to be executed. The offset is always relative to the base of the code segment (CS). The lower 16 bits (bits 0–15) of the EIP contain the 16-bit



NOTE:
0 indicates Intel Reserved: do not define; see Section 2.1.6.

Figure 2.2. Flags Register

instruction pointer named IP, which is used for 16-bit addressing.

2.1.1.3 Flags Register

The flags register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS control certain operations and indicate status of the Intel486 DX2 microprocessor. The lower 16 bits (bits 0–15) of EFLAGS contain the 16-bit register named FLAGS, which is most useful when executing 8086 and 80286 code. EFLAGS is shown in Figure 2.2.

EFLAGS bits 1, 3, 5, 15 and 19–31 are "undefined". When these bits are stored during interrupt processing or with a PUSHF instruction (push flags onto stack), a one is stored in bit 1 and zeros in bits 3, 5, 15 and 19–31.

The EFLAGS register in the Intel486 DX microprocessor contains a new bit not previously defined. The new bit, AC, is defined in the upper 16 bits of the register and it enables faults on accesses to misaligned data.

AC (Alignment Check, bit 18)

The AC bit enables the generation of faults if a memory reference is to a misaligned address. Alignment faults are enabled when AC is set to 1. A mis-aligned address is a word access to an odd address, a dword access to an address that is not on a dword boundary, or an 8-byte reference to an address that is not on a 64-bit word boundary. See Section 7.1.6 for more information on operand alignment.

Alignment faults are only generated by programs running at privilege level 3. The AC bit setting is ignored at privilege levels 0, 1 and 2. Note that references to the descriptor tables (for selector loads), or the task state segment (TSS), are implicitly level 0 references even if the instructions causing the references are executed at level 3. Alignment faults are reported through interrupt 17, with an error code of 0. Table 2.1 gives the alignment required for the Intel486 DX microprocessor data types.

Table 2.1. Data Type Alignment Requirements

Memory Access	Alignment (Byte Boundary)
Word	2
Dword	4
Single Precision Real	4
Double Precision Real	8
Extended Precision Real	8
Selector	2
48-Bit Segmented Pointer	4
32-Bit Flat Pointer	4
32-Bit Segmented Pointer	2
48-Bit "Pseudo-Descriptor"	4
FSTENV/FLDENV Save Area	4/2 (On Operand Size)
FSAVE/FRSTOR Save Area	4/2 (On Operand Size)
Bit String	4

IMPLEMENTATION NOTE:
Several instructions on the Intel486 DX microprocessor generate misaligned references, even if their memory address is aligned. For example, on the Intel486 DX microprocessor, the SGDT/SIDT (store global/interrupt descriptor table) instruction reads/writes two bytes, and then reads/writes four bytes from a "pseudo-descriptor" at the given address. The Intel486 DX microprocessor will generate misaligned references unless the address is on a 2 mod 4 boundary. The FSAVE and FRSTOR instructions (floating point save and restore state) will generate misaligned references for one-half of the register save/restore cycles. The Intel486 DX microprocessor will not cause any AC faults if the effective address given in the instruction has the proper alignment.

VM (Virtual 8086 Mode, bit 17)

The VM bit provides Virtual 8086 Mode within Protected Mode. If set while the Intel486 DX microprocessor is in Protected Mode, the Intel486 DX microprocessor will switch to Virtual 8086 operation, handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes. The VM bit can be set only in Protected Mode, by the IRET instruction (if current privilege level = 0) and by task switches at any privilege level. The VM bit is unaffected by POPF. PUSHF always pushes a 0 in this bit, even if executing in Virtual 8086 Mode. The EFLAGS image pushed during interrupt processing or saved during task switches will contain a 1 in this bit if the interrupted code was executing as a Virtual 8086 Task.

RF (Resume Flag, bit 16)

The RF flag is used in conjunction with the debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. When RF is set, it causes any debug fault to be ignored on the next instruction. RF is then automatically reset at the successful completion of every instruction (no faults are signalled) except the IRET instruction, the POPF instruction, (and JMP, CALL, and INT instructions causing a task switch). These instructions set RF to the value specified by the memory image. For example, at the end of the breakpoint service routine, the IRET instruction can pop an EFLAG image having the RF bit set and resume the program's execution at the breakpoint address without generating another breakpoint fault on the same location.

NT (Nested Task, bit 14)

This flag applies to Protected Mode. NT is set to indicate that the execution of this task is nested within another task. If set, it indicates that the current nested task's Task State Segment (TSS) has a valid back link to the previous task's TSS. This bit is set or reset by control transfers to other tasks. The value of NT in EFLAGS is tested by the IRET instruction to determine whether to do an inter-task return or an intra-task return. A POPF or an IRET instruction **will** affect the setting of this bit according to the image popped, at any privilege level.

IOPL (Input/Output Privilege Level, bits 12-13)

This two-bit field applies to Protected Mode. IOPL indicates the numerically maximum CPL (current privilege level) value permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O Permission Bitmap. It also indicates the maximum CPL value allowing alteration of the IF (INTR Enable Flag) bit when new values are popped into the EFLAG register. POPF and IRET instruction can alter the IOPL field when executed at CPL = 0. Task switches can always alter the IOPL field, when the new flag image is loaded from the incoming task's TSS.

OF (Overflow Flag, bit 11)

OF is set if the operation resulted in a signed overflow. Signed overflow occurs when the operation resulted in carry/borrow **into** the sign bit (high-order bit) of the result but did not result in a carry/borrow **out of** the high-order bit, or vice-versa. For 8-, 16-, 32-bit operations, OF is set according to overflow at bit 7, 15, 31, respectively.

DF (Direction Flag, bit 10)

DF defines whether ESI and/or EDI registers postdecrement or postincrement during the string instructions. Postincrement occurs if DF is reset. Postdecrement occurs if DF is set.

IF (INTR Enable Flag, bit 9)

The IF flag, when set, allows recognition of external interrupts signalled on the INTR pin. When IF is reset, external interrupts signalled on the INTR are not recognized. IOPL indicates the maximum CPL value allowing alteration of the IF bit when new values are popped into EFLAGS or FLAGS.

TF (Trap Enable Flag, bit 8)

TF controls the generation of exception 1 trap when single-stepping through code. When TF is set, the Intel486 DX microprocessor generates an exception 1 trap after the next instruction is executed. When TF is reset, exception 1 traps occur only as a function of the breakpoint addresses loaded into debug registers DR0–DR3.

SF (Sign Flag, bit 7)

SF is set if the high-order bit of the result is set, it is reset otherwise. For 8-, 16-, 32-bit operations, SF reflects the state of bit 7, 15, 31 respectively.

ZF (Zero Flag, bit 6)

ZF is set if all bits of the result are 0. Otherwise it is reset.

AF (Auxiliary Carry Flag, bit 4)

The Auxiliary Flag is used to simplify the addition and subtraction of packed BCD quantities. AF is set if the operation resulted in a carry out of bit 3 (addition) or a borrow into bit 3 (subtraction). Otherwise AF is reset. AF is affected by carry out of, or borrow into bit 3 only, regardless of overall operand length: 8, 16 or 32 bits.

PF (Parity Flags, bit 2)

PF is set if the low-order eight bits of the operation contains an even number of "1's" (even parity). PF is reset if the low-order eight bits have odd parity. PF is a function of only the low-order eight bits, regardless of operand size.

CF (Carry Flag, bit 0)

CF is set if the operation resulted in a carry out of (addition), or a borrow into (subtraction) the high-order bit. Otherwise CF is reset. For 8-, 16- or 32-bit operations, CF is set according to carry/borrow at bit 7, 15 or 31, respectively.

NOTE:
In these descriptions, "set" means "set to 1," and "reset" means "reset to 0."

2.1.1.4 Segment Registers

Six 16-bit segment registers hold segment selector values identifying the currently addressable memory segments. In protected mode, each segment may range in size from one byte up to the entire linear and physical address space of the machine, 4 Gbytes (2^{32} bytes). In real address mode, the maximum segment size is fixed at 64 Kbytes (2^{16} bytes).

The six addressable segments are defined by the segment registers CS, SS, DS, ES, FS and GS. The selector in CS indicates the current code segment; the selector in SS indicates the current stack segment; the selectors in DS, ES, FS and GS indicate the current data segments.

2.1.1.5 Segment Descriptor Cache Registers

The segment descriptor cache registers are not programmer visible, yet it is very useful to understand their content. A programmer invisible descriptor cache register is associated with each programmer-visible segment register, as shown by Figure 2.3. Each descriptor cache register holds a 32-bit base address, a 32-bit segment limit, and the other necessary segment attributes.

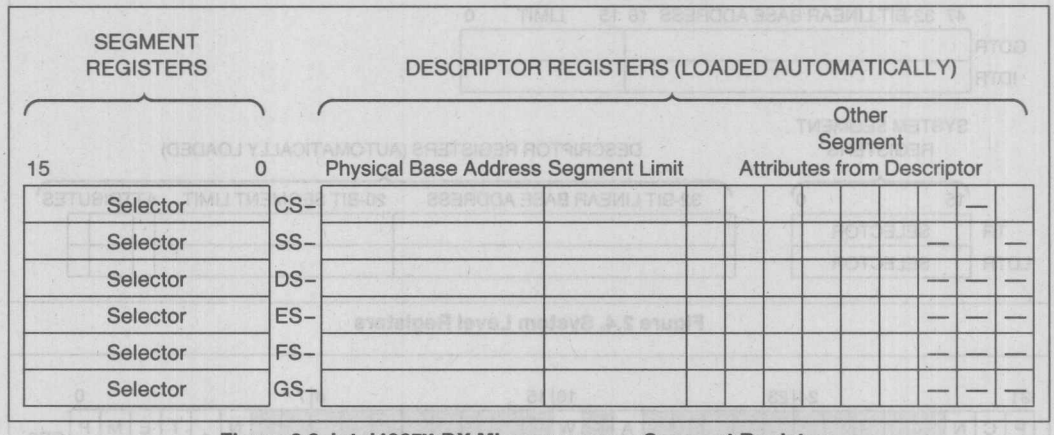


Figure 2.3. Intel486™ DX Microprocessor Segment Registers and Associated Descriptor Cache Registers

When a selector value is loaded into a segment register, the associated descriptor cache register is automatically updated with the correct information. In Real Address Mode, only the base address is updated directly (by shifting the selector value four bits to the left), since the segment maximum limit and attributes are fixed in Real Mode. In Protected Mode, the base address, the limit, and the attributes are all updated per the contents of the segment descriptor indexed by the selector.

Whenever a memory reference occurs, the segment descriptor cache register associated with the segment being used is automatically involved with the memory reference. The 32-bit segment base address becomes a component of the linear address calculation, the 32-bit limit is used for the limit-check operation, and the attributes are checked against the type of memory reference requested.

2.1.2 SYSTEM LEVEL REGISTERS

The system level registers, Figure 2.4, control operation of the on-chip cache, the on-chip floating point unit (FPU) and the segmentation and paging mechanisms. These registers are only accessible to programs running at privilege level 0, the highest privilege level.

The system level registers include three control registers and four segmentation base registers. The three control registers are CR0, CR2 and CR3. CR1 is reserved for future Intel processors. The four segmentation base registers are the Global Descriptor Table Register (GDTR), the Interrupt Descriptor Table Register (IDTR), the Local Descriptor Table Register (LDTR) and the Task State Segment Register (TR).

2.1.2.1 Control Registers

Control Register 0 (CR0)

CR0, shown in Figure 2.5, contains 10 bits for control and status purposes. Five of the bits defined in the Intel486 DX microprocessor's CR0 are newly defined. The new bits are CD, NW, AM, WP and NE. The function of the bits in CR0 can be categorized as follows:

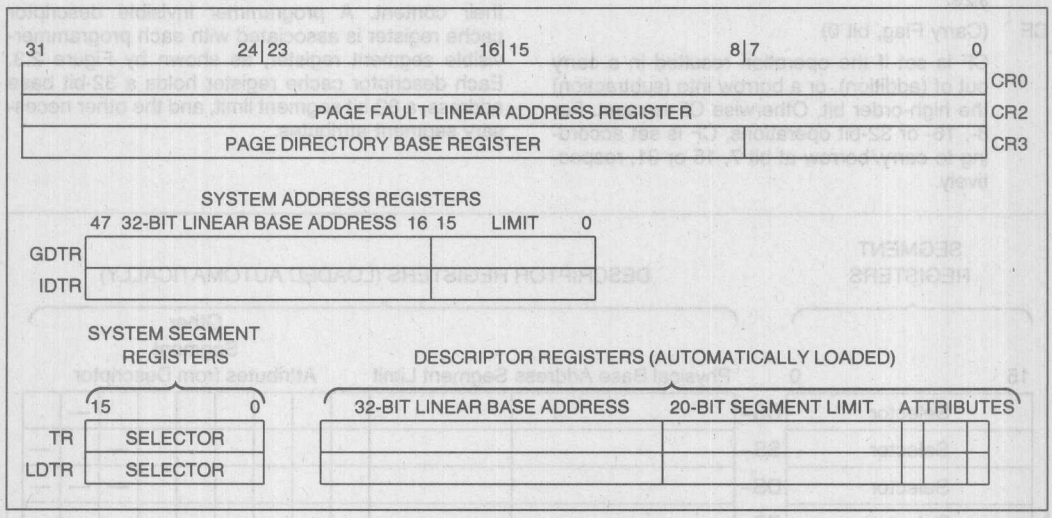


Figure 2.4. System Level Registers

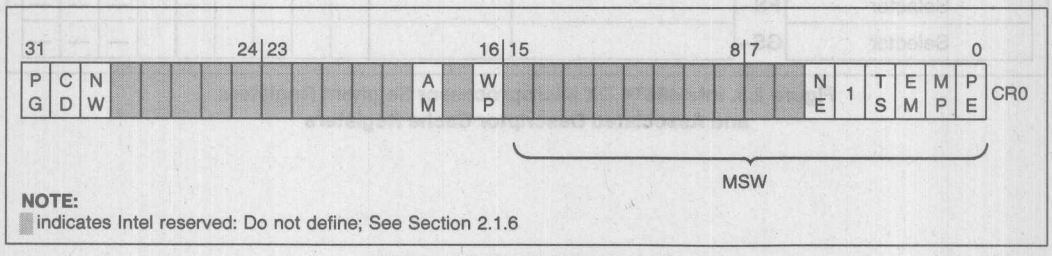


Figure 2.5. Control Register 0

Intel486 DX Microprocessor Operating Modes: PG, PE (Table 2.2)

On-Chip Cache Control Modes: CD, NW (Table 2.3)

On-Floating Point Unit Control: TS, EM, MP, NE (Table 2.4)

Alignment Check Control: AM

Supervisor Write Protect: WP

Table 2.2. Processor Operating Modes

PG	PE	Mode
0	0	REAL Mode. Exact 8086 semantics, with 32-bit extensions available with prefixes.
0	1	Protected Mode. Exact 80286 semantics, plus 32-bit extensions through both prefixes and "default" prefix setting associated with code segment descriptors. Also, a sub-mode is defined to support a virtual 8086 within the context of the extended 80286 protection model.
1	0	UNDEFINED. Loading CR0 with this combination of PG and PE bits will raise a GP fault with error code 0.
1	1	Paged Protected Mode. All the facilities of Protected mode, with paging enabled underneath segmentation.

Table 2.3. On-Chip Cache Control Modes

CD	NW	Operating Mode
1	1	Cache fills disabled, write-through and invalidates disabled.
1	0	Cache fills disabled, write-through and invalidates enabled.
0	1	INVALID. If CR0 is loaded with this configuration of bits, a GP fault with error code is raised.
0	0	Cache fills enabled, write-through and invalidates enabled.

Table 2.4. On-Chip Floating Point Unit Control

CR0 BIT			Instruction Type	
EM	TS	MP	Floating-Point	Wait
0	0	0	Execute	Execute
0	0	1	Execute	Execute
0	1	0	Trap 7	Execute
0	1	1	Trap 7	Trap 7
1	0	0	Trap 7	Execute
1	0	1	Trap 7	Execute
1	1	0	Trap 7	Execute
1	1	1	Trap 7	Trap 7

The low-order 16 bits of CR0 are also known as the Machine Status Word (MSW), for compatibility with the 80286 protected mode. LMSW and SMSW (load and store MSW) instructions are taken as special aliases of the load and store CR0 operations, where only the low-order 16 bits of CR0 are involved. The LMSW and SMSW instructions in the Intel486 DX microprocessor work in an identical fashion to the LMSW and SMSW instructions in the 80286 (i.e., they only operate on the low-order 16 bits of CR0 and ignores the new bits). New Intel486 DX microprocessor operating systems should use the MOV CR0, Reg instruction.

The defined CR0 bits are described below.

PG (Paging Enable, bit 31)

The PG bit is used to indicate whether paging is enabled (PG = 1) or disabled (PG = 0). See Table 2.2.

CD (Cache Disable, bit 30)

The CD bit is used to enable the on-chip cache. When CD = 1, the cache will not be filled on cache misses. When CD = 0, cache fills may be performed on misses. See Table 2.3.

The state of the CD bit, the cache enable input pin (KEN#), and the relevant page cache disable (PCD) bit determine if a line read in response to a cache miss will be installed in the cache. A line is installed in the cache only if CD = 0 and KEN# and PCD are both zero. The relevant PCD bit comes from either the page table entry, page directory entry or control register 3. Refer to Section 5.6 for more details on page cacheability.

CD is set to one after RESET.

NW (Not Write-Through, bit 29)

The NW bit enables on-chip cache write-throughs and write-invalidate cycles (NW = 0). When NW = 0, all writes, including cache hits, are sent out to the pins. Invalidate cycles are enabled when NW = 0. During an invalidate cycle a line will be removed from the cache if the invalidate address hits in the cache. See Table 2.3.

When NW = 1, write-throughs and write-invalidate cycles are disabled. A write will not be sent to the pins if the write hits in the cache. With NW = 1 the only write cycles that reach the external bus are cache misses. Write hits with NW = 1 will never update main memory. Invalidate cycles are ignored when NW = 1.

AM (Alignment Mask, bit 18)

The AM bit controls whether the alignment check (AC) bit in the flag register (EFLAGS) can allow an alignment fault. AM = 0 disables the AC bit. AM = 1 enables the AC bit. AM = 0 is the Intel386 microprocessor compatible mode.

Intel386 microprocessor software may load incorrect data into the AC bit in the EFLAGS register. Setting AM = 0 will prevent AC faults from occurring before the Intel486 DX microprocessor has created the AC interrupt service routine.

WP (Write Protect, bit 16)

WP protects read-only pages from supervisor write access. The Intel386 microprocessor allows a read-only page to be written from privilege levels 0–2. The Intel486 DX microprocessor is compatible with the Intel386 microprocessor when WP = 0. WP = 1 forces a fault on a write to a read-only page from any privilege level. Operating systems with Copy-on-Write features can be supported with the WP bit. Refer to Section 4.5.3 for further details on use of the WP bit.

NE (Numerics Exception, bit 5)

The NE bit controls whether unmasked floating point exceptions (UFPE) are handled through interrupt vector 16 (NE = 1) or through an external interrupt (NE = 0). NE = 0 (default at reset) supports the DOS operating system error reporting scheme from the 8087, 80287 and Intel387 math coprocessor. In DOS systems, math coprocessor errors are reported via external interrupt vector 13. DOS uses interrupt vector 16 for an operating system call. Refer to Sections 6.2.13 and 7.2.14 for more information on floating point error reporting.

For any UFPE the floating point error output pin (FERR#) will be driven active.

For NE = 0, the Intel486 DX microprocessor works in conjunction with the ignore numeric error input (IGNNE#) and the FERR# output pins. When a UFPE occurs and the IGNNE# input is inactive, the Intel486 DX microprocessor freezes immediately before executing the next floating point instruction. An external interrupt controller will supply an interrupt vector when FERR# is driven active. The UFPE is ignored if IGNNE# is active and floating point execution continues.

NOTE:

The freeze does not take place if the next instruction is one of the control instructions FNCLEX, FNINIT, FNSAVE, FNSTENV, FNSTCW, FNSTSW, FNSTSW AX, FNENI, FNDISI and FNSETPM. The freeze does occur if the next instruction is WAIT.

For NE = 1, any UFPE will result in a software interrupt 16, immediately before executing the next non-control floating point or WAIT instruction. The ignore numeric error input (IGNNE#) signal will be ignored.

TS (Task Switched, bit 3)

The TS bit is set whenever a task switch operation is performed. Execution of a floating point instruction with TS = 1 will cause a device not available (DNA) fault (trap vector 7). If TS = 1 and MP = 1 (monitor coprocessor in CR0) a WAIT instruction will cause a DNA fault. See Table 2.4.

EM (Emulate Coprocessor, bit 2)

The EM bit determines whether floating point instructions are trapped (EM = 1) or executed. If EM = 1, all floating point instructions will cause fault 7.

NOTE:

WAIT instructions are not affected by the state of EM. See Table 2.4.

MP (Monitor Coprocessor, bit 1)

The MP bit is used in conjunction with the TS bit to determine if WAIT instructions should trap. If MP = 1 and TS = 1, WAIT instructions cause fault 7. Refer to Table 2.4. The TS bit is set to 1 on task switches by the Intel486 DX microprocessor. Floating point instructions are not affected by the state of the MP bit. It is recommended that the MP bit be set to one for the normal operation of the Intel486 DX microprocessor.

PE (Protection Enable, bit 0)

The PE bit enables the segment based protection mechanism. If PE = 1 protection is enabled. When PE = 0 the Intel486 DX microprocessor operates in REAL mode, with segment based protection disabled, and addresses formed as in an 8086. Refer to Table 2.2.

All new CR0 bits added to the Intel386 and Intel486 DX microprocessors, except for ET and NE, are upward compatible with the 80286 because they are in register bits not defined in the 80286. For strict compatibility with the 80286, the load machine status word (LMSW) instruction is defined to not change the ET or NE bits.

Control Register 1 (CR1)

CR1 is reserved for use in future Intel microprocessors.

Control Register 2 (CR2)

CR2, shown in Figure 2.6, holds the 32-bit linear address that caused the last page fault detected. The error code pushed onto the page fault handler's stack when it is invoked provides additional status information on this page fault.

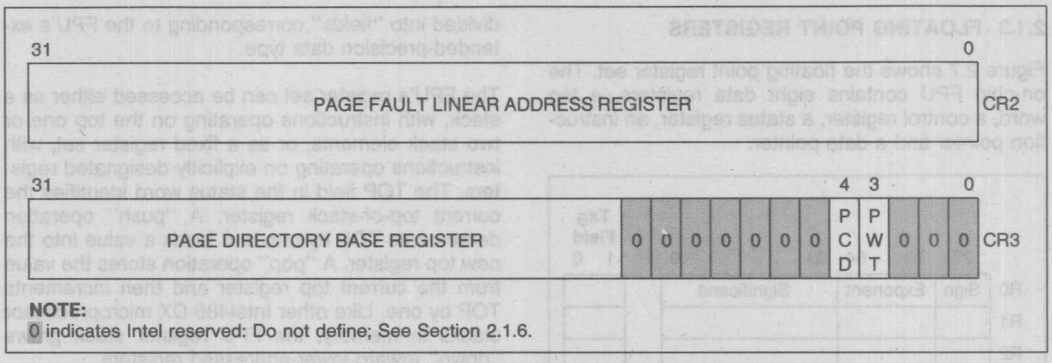


Figure 2.6. Control Registers 2 and 3

Control Register 3 (CR3)

CR3, shown in Figure 2.6, contains the physical base address of the page directory table. The Intel486 DX microprocessor page directory is always page aligned (4 Kbyte-aligned). This alignment is enforced by only storing bits 20–31 in CR3.

In the Intel486 DX microprocessor CR3 contains two new bits, page write-through (PWT) (bit 3) and page cache disable (PCD) (bit 4). The page table entry (PTE) and page directory entry (PDE) also contain PWT and PCD bits. PWT and PCD control page cacheability. When a page is accessed in external memory, the state of PWT and PCD are driven out on the PWT and PCD pins. The source of PWT and PCD can be CR3, the PTE or the PDE. PWT and PCD are sourced from CR3 when the PDE is being updated. When paging is disabled (PG = 0 in CR0), PCD and PWT are assumed to be 0, regardless of their state in CR3.

A task switch through a task state segment (TSS) which changes the values in CR3, or an explicit load into CR3 with any value, will invalidate all cached page table entries in the translation lookaside buffer (TLB).

The page directory base address in CR3 is a physical address. The page directory can be paged out while its associated task is suspended, but the operating system must ensure that the page directory is resident in physical memory before the task is dispatched. The entry in the TSS for CR3 has a physical address, with no provision for a present bit. This means that the page directory for a task must be resident in physical memory. The CR3 image in a TSS must point to this area, before the task can be dispatched through its TSS.

2.1.2.2 System Address Registers

Four special registers are defined to reference the tables or segments supported by the 80286, Intel386 and Intel486 DX microprocessor protection model. These tables or segments are:

GDT (Global Descriptor Table)
IDT (Interrupt Descriptor Table)
LDT (Local Descriptor Table)
TSS (Task State Segment)

The addresses of these tables and segments are stored in special registers, the System Address and System Segment Registers, illustrated in Figure 2.4. These registers are named GDTR, IDTR, LDTR and TR respectively. Section 4, Protected Mode Architecture, describes the use of these registers.

System Address Registers: GDTR and IDTR

The GDTR and IDTR hold the 32-bit linear base address and 16-bit limit of the GDT and IDT, respectively.

Since the GDT and IDT segments are global to all tasks in the system, the GDT and IDT are defined by 32-bit linear addresses (subject to page translation if paging is enabled) and 16-bit limit values.

System Segment Registers: LDTR and TR

The LDTR and TR hold the 16-bit selector for the LDT descriptor and the TSS descriptor, respectively.

Since the LDT and TSS segments are task specific segments, the LDT and TSS are defined by selector values stored in the system segment registers.

NOTE:
A programmer-invisible segment descriptor register is associated with each system segment register.

2.1.3 FLOATING POINT REGISTERS

Figure 2.7 shows the floating point register set. The on-chip FPU contains eight data registers, a tag word, a control register, a status register, an instruction pointer and a data pointer.

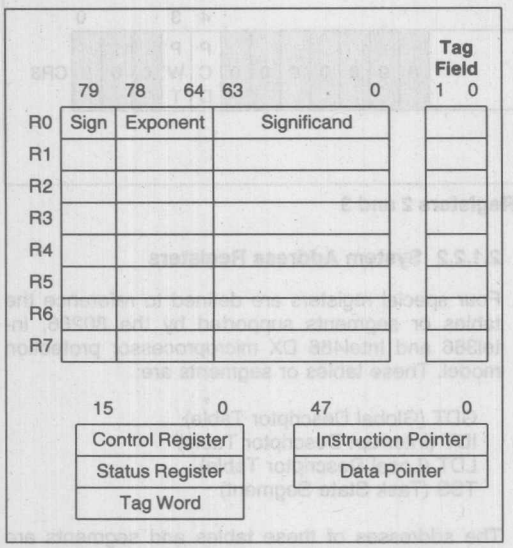


Figure 2.7. Floating Point Registers

The operation of the Intel486 DX microprocessor's on-chip floating point unit is exactly the same as the Intel387 math coprocessor. Software written for the Intel387 math coprocessor will run on the on-chip floating point unit (FPU) without any modifications.

2.1.3.1 Data Registers

Floating point computations use the Intel486 DX microprocessor's FPU data registers. These eight 80-bit registers provide the equivalent capacity of twenty 32-bit registers. Each of the eight data registers is divided into "fields" corresponding to the FPU's extended-precision data type.

The FPU's register set can be accessed either as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers. The TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by one and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by one. Like other Intel486 DX microprocessor stacks in memory, the FPU register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to use. This explicit register addressing is also relative to TOP.

2.1.3.2 Tag Word

The tag word marks the content of each numeric data register, as shown in Figure 2.8. Each two-bit tag represents one of the eight data registers. The principal function of the tag word is to optimize the FPUs performance and stack handling by making it possible to distinguish between empty and nonempty register locations. It also enables exception handlers to check the contents of a stack location without the need to perform complex decoding of the actual data.

2.1.3.3 Status Word

The 16-bit status word reflects the overall state of the FPU. The status word is shown in Figure 2.9 and is located in the status register.

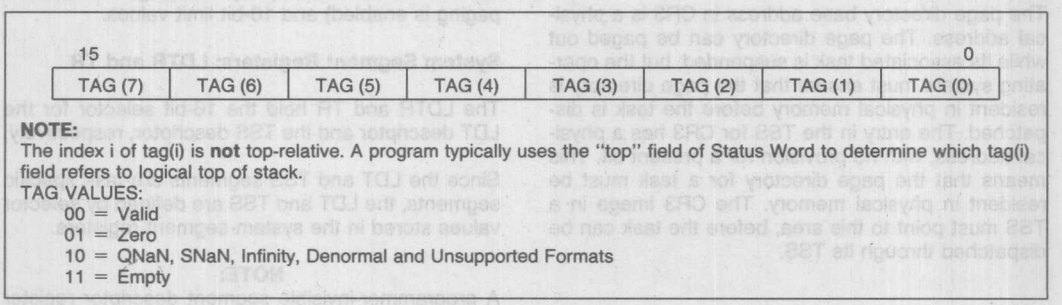


Figure 2.8. FPU Tag Word

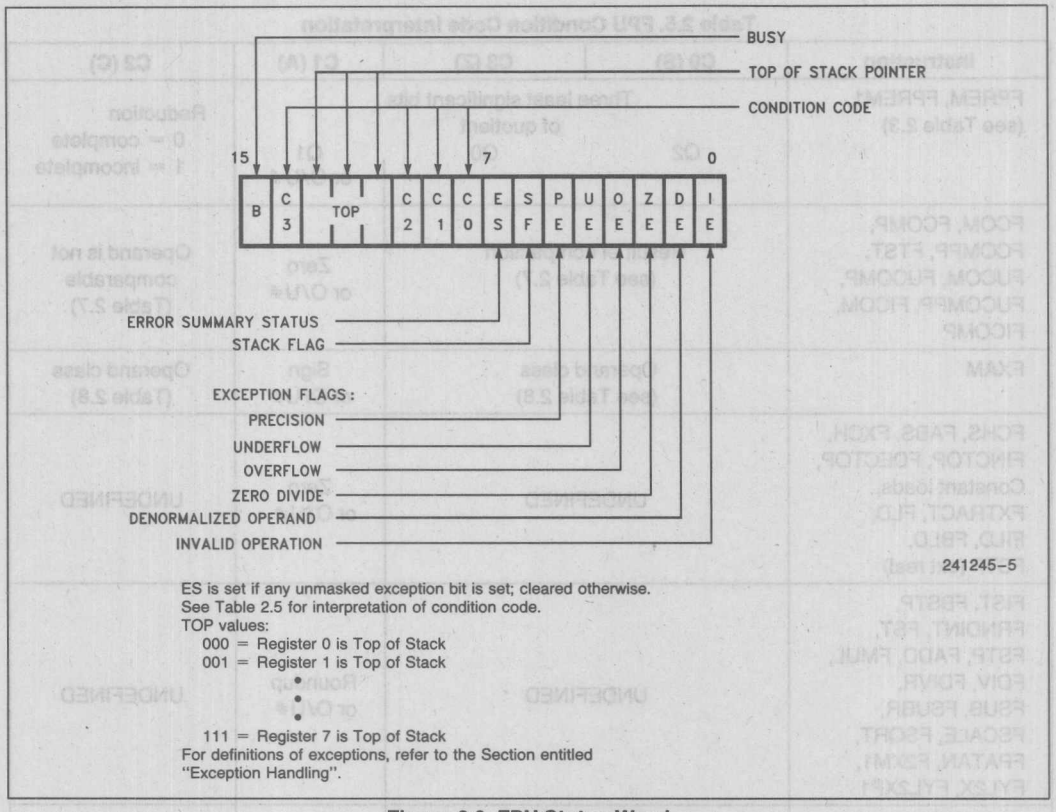


Figure 2.9. FPU Status Word

The B bit (Busy, bit 15) is included for 8087 compatibility. The B bit reflects the contents of the ES bit (bit 7 of the status word).

Bits 13–11 (TOP) point to the FPU register that is the current top-of-stack.

The four numeric condition code bits, C0–C3, are similar to the flags in EFLAGS. Instructions that perform arithmetic operations update C0–C3 to reflect the outcome. The effects of these instructions on the condition codes are summarized in Tables 2.5 through 2.8.

Table 2.5. FPU Condition Code Interpretation

Instruction	C0 (S)	C3 (Z)	C1 (A)	C2 (C)
FPREM, FPREM1 (see Table 2.3)	Three least significant bits of quotient Q2	Q0	Q1 or O/U#	Reduction 0 = complete 1 = incomplete
FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of comparison (see Table 2.7)		Zero or O/U#	Operand is not comparable (Table 2.7)
FXAM	Operand class (see Table 2.8)		Sign or O/U#	Operand class (Table 2.8)
FCHS, FABS, FXCH, FINCTOP, FDECTOP, Constant loads, FXTRACT, FLD, FILD, FBLD, FSTP (ext real)	UNDEFINED		Zero or O/U#	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/U#	UNDEFINED
FPTAN, FSIN FCOS, FSINCOS	UNDEFINED		Roundup or O/U#, undefined if C2 = 1	Reduction 0 = complete 1 = incomplete
FLDENV, FRSTOR	Each bit loaded from memory			
FINIT	Clears these bits			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FSAVE	UNDEFINED			

O/U#	When both IE and SF bits of status word are set, indicating a stack exception, this bit distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).
Reduction	If FPREM or FPREM1 produces a remainder that is less than the modulus, reduction is complete. When reduction is incomplete the value at the top of the stack is a partial remainder, which can be used as input to further reduction. For FPTAN, FSIN, FCOS, and FSINCOS, the reduction bit is set if the operand at the top of the stack is too large. In this case the original operand remains at the top of the stack.
Roundup	When the PE bit of the status word is set, this bit indicates whether the last rounding in the instruction was upward.
UNDEFINED	Do not rely on finding any specific value in these bits.

Table 2.6. Condition Code Interpretation after FPREM and FPREM1 Instructions

Condition Code				Interpretation after FPREM and FPREM1	
C2	C3	C1	C0		
1	X	X	X	Incomplete Reduction: further interaction required for complete reduction	
	Q1	Q0	Q2	Q MOD8	
	0	0	0	0	
	0	1	0	1	
	1	0	0	2	
0	1	1	0	3	Complete Reduction: C0, C3, C1 contain three least significant bits of quotient
	0	0	1	4	
	0	1	1	5	
	1	0	1	6	
	1	1	1	7	

Table 2.7. Condition Code Resulting from Comparison

Order	C3	C2	C0
TOP > Operand	0	0	0
TOP < Operand	0	0	1
TOP = Operand	1	0	0
Unordered	1	1	1

Table 2.8. Condition Code Defining Operand Class

C3	C2	C1	C0	Value at TOP
0	0	0	0	+ Unsupported
0	0	0	1	+ NaN
0	0	1	0	− Unsupported
0	0	1	1	− NaN
0	1	0	0	+ Normal
0	1	0	1	+ Infinity
0	1	1	0	− Normal
0	1	1	1	− Infinity
1	0	0	0	+ 0
1	0	0	1	+ Empty
1	0	1	0	− 0
1	0	1	1	− Empty
1	1	0	0	+ Denormal
1	1	1	0	− Denormal

Bit 7 is the error summary (ES) status bit. The ES bit is set if any unmasked exception bit (bits 0–5 in the status word) is set; ES is clear otherwise. The FERR# (floating point error) signal is asserted when ES is set.

Bit 6 is the stack flag (SF). This bit is used to distinguish invalid operations due to stack overflow or underflow. When SF is set, bit 9 (C1) distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).

Table 2.9 shows the six exception flags in bits 0–5 of the status word. Bits 0–5 are set to indicate that the FPU has detected an exception while executing an instruction.

The six exception flags in the status word can be individually masked by mask bits in the FPU control word. Table 2.9 lists the exception conditions, and their causes in order of precedence. Table 2.9 also shows the action taken by the FPU if the corresponding exception flag is masked.

An exception that is not masked by the control word will cause three things to happen: the corresponding exception flag in the status word will be set, the ES bit in the status word will be set and the FERR# output signal will be asserted. When the Intel486 DX microprocessor attempts to execute another floating point or WAIT instruction, exception 16 occurs or an external interrupt happens if the NE = 1 in control register 0. The exception condition must be resolved via an interrupt service routine. The FPU saves the address of the floating point instruction that caused the exception and the address of any memory operand required by that instruction in the instruction and data pointers (see Section 2.1.3.4).

Note that when a new value is loaded into the status word by the FLDENV (load environment) or FRSTOR (restore state) instruction, the value of ES (bit 7) and its reflection in the B bit (bit 15) are not derived from the values loaded from memory. The values of ES and B are dependent upon the values of the exception flags in the status word and their corresponding masks in the control word. If ES is set in such a case, the FERR# output of the Intel486 DX microprocessor is activated immediately.

2.1.3.4 Instruction and Data Pointers

Because the FPU operates in parallel with the ALU (in the Intel486 DX and Intel486 microprocessors the arithmetic and logic unit (ALU) consists of the base architecture registers), any errors detected by the FPU may be reported after the ALU has executed the floating point instruction that caused it. To allow identification of the failing numeric instruction, the Intel486 DX microprocessor contains two pointer registers that supply the address of the failing numeric instruction and the address of its numeric memory operand (if appropriate).

Table 2.9. FPU Exceptions

Exception	Cause	Default Action (if exception is masked)
Invalid Operation	Operation on a signaling NaN, unsupported format, indeterminate form (0 * ∞, 0/0, (+ ∞) + (− ∞), etc.), or stack overflow/underflow (SF is also set).	Result is a quiet NaN, integer indefinite, or BCD indefinite
Denormalized Operand	At least one of the operands is denormalized, i.e., it has the smallest exponent but a nonzero significand.	Normal processing continues
Zero Divisor	The divisor is zero while the dividend is a noninfinite, nonzero number.	Result is ∞
Overflow	The result is too large in magnitude to fit in the specified format.	Result is largest finite value or ∞
Underflow	The true result is nonzero but too small to be represented in the specified format, and, if underflow exception is masked, denormalization causes loss of accuracy.	Result is denormalized or zero
Inexact Result (Precision)	The true result is not exactly representable in the specified format (e.g., 1/3); the result is rounded according to the rounding mode.	Normal processing continues

The instruction and data pointers are provided for user-written error handlers. These registers are accessed by the FLDENV (load environment), FSTENV (store environment), FSAVE (save state) and FRSTOR (restore state) instructions. Whenever the Intel486 DX microprocessor decodes a new floating point instruction, it saves the instruction (including any prefixes that may be present), the address of the operand (if present) and the opcode.

The instruction and data pointers appear in one of four formats depending on the operating mode of the Intel486 DX microprocessor (protected mode or real-address mode) and depending on the operand-size attribute in effect (32-bit operand or 16-bit operand). When the Intel486 DX microprocessor is in the virtual-86 mode, the real address mode formats are used. The four formats are shown in Figures 2.10–2.13. The floating point instructions FLDENV, FSTENV, FSAVE and FRSTOR are used to transfer these values to and from memory. Note that the value of the data pointer is undefined if the prior floating point instruction did not have a memory operand.

NOTE:
The operand size attribute is the D bit in a segment descriptor.

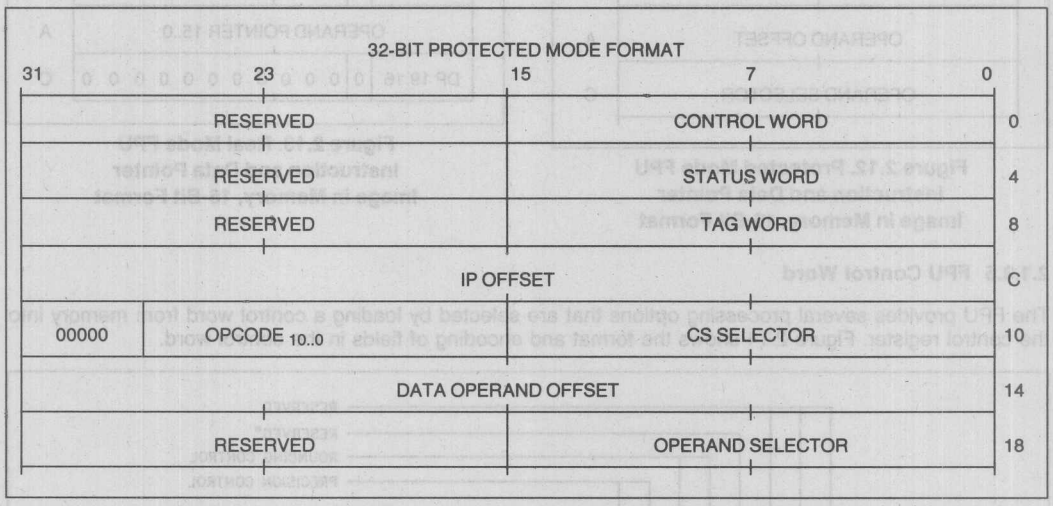


Figure 2.10. Protected Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

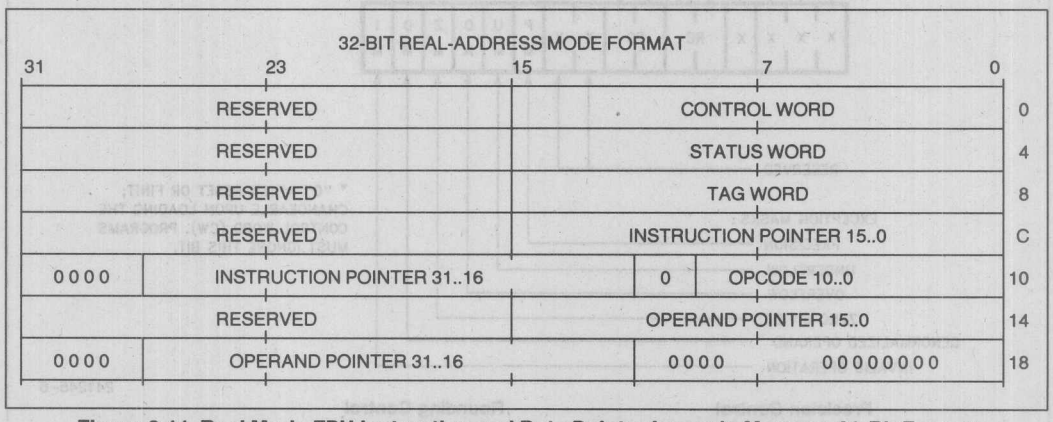


Figure 2.11. Real Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

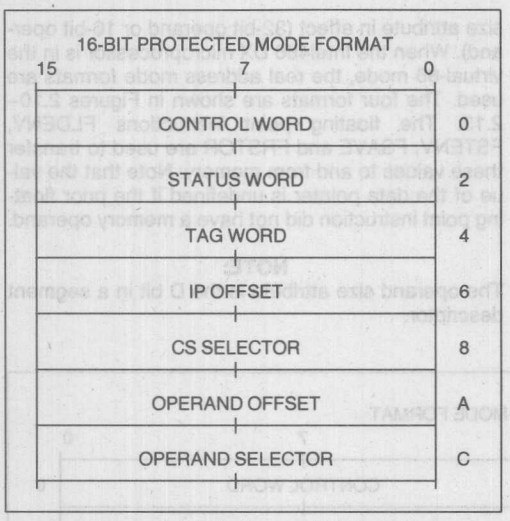


Figure 2.12. Protected Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

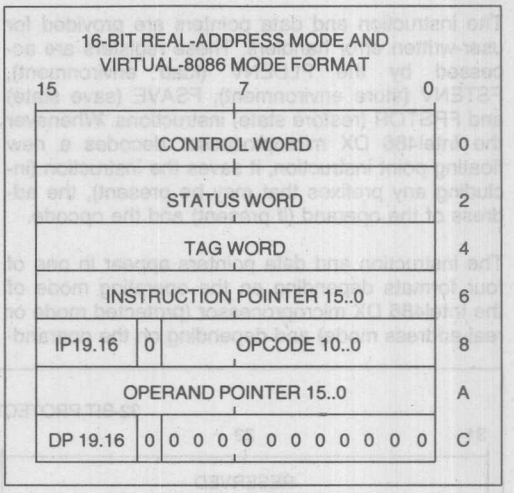


Figure 2.13. Real Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

2.1.3.5 FPU Control Word

The FPU provides several processing options that are selected by loading a control word from memory into the control register. Figure 2.14 shows the format and encoding of fields in the control word.

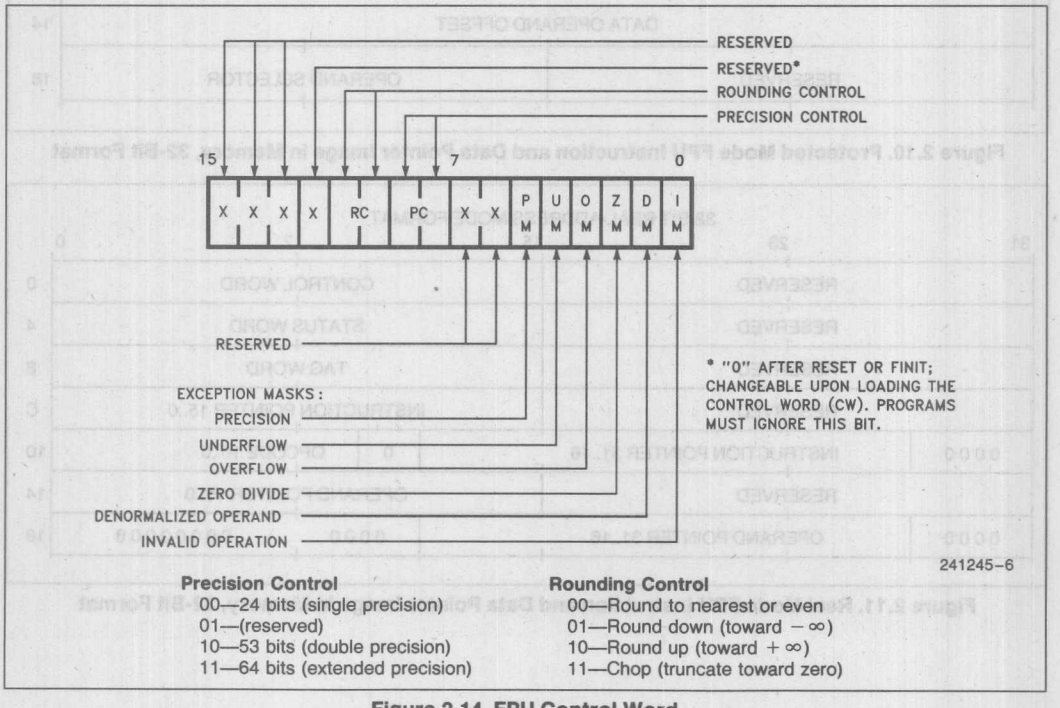


Figure 2.14. FPU Control Word

The low-order byte of the FPU control word configures the FPU error and exception masking. Bits 0–5 of the control word contain individual masks for each of the six exceptions that the FPU recognizes.

The high-order byte of the control word configures the FPU operating mode, including precision and rounding.

RC (Rounding Control, bits 10–11)

The RC bits provide for directed rounding and true chop, as well as the unbiased round to nearest even mode specified in the IEEE standard. Rounding control affects only those instructions that perform rounding at the end of the operation (and thus can generate a precision exception); namely, FST, FSTP, FIST, all arithmetic instructions (except FPREM, FPREM1, FXTRACT, FABS and FCHS), and all transcendental instructions.

PC (Precision Control, bits 8–9)

The PC bits can be used to set the FPU internal operating precision of the significand at less than the default of 64 bits (extended precision). This can be useful in providing compatibility with early generation arithmetic processors of smaller precision. PC affects only the instructions ADD, SUB, DIV, MUL, and SQRT. For all other instructions, either the precision is determined by the opcode or extended precision is used.

2.1.4 DEBUG AND TEST REGISTERS

2.1.4.1 Debug Registers

The six programmer accessible debug registers, Figure 2.15, provide on-chip support for debugging. Debug registers DR0–3 specify the four linear breakpoints. The Debug control register DR7, is used to set the breakpoints and the Debug Status Register, DR6, displays the current state of the breakpoints. The use of the Debug registers is described in Section 9.

Debug Registers

LINEAR BREAKPOINT ADDRESS 0	DR0
LINEAR BREAKPOINT ADDRESS 1	DR1
LINEAR BREAKPOINT ADDRESS 2	DR2
LINEAR BREAKPOINT ADDRESS 3	DR3
Intel Reserved Do Not Define	DR4
Intel Reserved Do Not Define	DR5
BREAKPOINT STATUS	DR6
BREAKPOINT CONTROL	DR7

Test Registers

CACHE TEST DATA	TR3
CACHE TEST STATUS	TR4
CACHE TEST CONTROL	TR5
TLB TEST CONTROL	TR6
TLB TEST STATUS	TR7

TLB = Translation Lookaside Buffer

Figure 2.15

2.1.4.2 Test Registers

The Intel486 DX microprocessor contains five test registers. The test registers are shown in Figure 2.15. TR6 and TR7 are used to control the testing of the translation lookaside buffer. TR3, TR4 and TR5 are used for testing the on-chip cache. The use of the test registers is discussed in Section 8.

2.1.5 REGISTER ACCESSIBILITY

There are a few differences regarding the accessibility of the registers in Real and Protected Mode. Table 2.10 summarizes these differences. See Section 4, Protected Mode Architecture, for further details.

Table 2.10. Register Usage

Register	Use in Real Mode		Use in Protected Mode		Use in Virtual 8086 Mode	
	Load	Store	Load	Store	Load	Store
General Registers	Yes	Yes	Yes	Yes	Yes	Yes
Segment Register	Yes	Yes	Yes	Yes	Yes	Yes
Flag Register	Yes	Yes	Yes	Yes	IOPL	IOPL*
Control Registers	Yes	Yes	PL = 0	PL = 0	No	Yes
GDTR	Yes	Yes	PL = 0	Yes	No	Yes
IDTR	Yes	Yes	PL = 0	Yes	No	Yes
LDTR	No	No	PL = 0	Yes	No	No
TR	No	No	PL = 0	Yes	No	No
FPU Data Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Control Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Status Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Instruction Pointer	Yes	Yes	Yes	Yes	Yes	Yes
FPU Data Pointer	Yes	Yes	Yes	Yes	Yes	Yes
Debug Registers	Yes	Yes	PL = 0	PL = 0	No	No
Test Registers	Yes	Yes	PL = 0	PL = 0	No	No

NOTES:
PL = 0: The registers can be accessed only when the current privilege level is zero.
*IOPL: The PUSHF and POPF instructions are made I/O Privilege Level sensitive in Virtual 86 Mode.

2.1.6 COMPATIBILITY

VERY IMPORTANT NOTE:
COMPATIBILITY WITH FUTURE PROCESSORS

In the preceding register descriptions, note certain Intel486 microprocessor register bits are Intel reserved. When reserved bits are called out, treat them as fully undefined. This is essential for your software compatibility with future processors! Follow the guidelines below:

1) **Do not depend on the states of any undefined bits when testing the values of defined register bits. Mask them out when testing.**
2) **Do not depend on the states of any undefined bits when storing them to memory or another register.**
3) **Do not depend on the ability to retain information written into any undefined bits.**
4) **When loading registers always load the undefined bits as zeros.**
5) **However, registers which have been previously stored may be reloaded without masking.**

Depending upon the values of undefined register bits will make your software dependent upon the unspecified Intel486 DX microprocessor handling of these bits. Depending on undefined values risks making your software incompatible with future processors that define usages for the Intel486 microprocessor-undefined bits. AVOID ANY SOFTWARE DEPENDENCE UPON THE STATE OF UNDEFINED Intel486 MICROPROCESSOR REGISTER BITS.

2.2 Instruction Set

The Intel486 DX microprocessor instruction set can be divided into 11 categories of operations:

Data Transfer
Arithmetic
Shift/Rotate
String Manipulation
Bit Manipulation
Control Transfer
High Level Language Support
Operating System Support
Processor Control
Floating Point
Floating Point Control

The Intel486 DX microprocessor instructions are listed in Section 10. Note that all floating point unit instruction mnemonics begin with an F.

All Intel486 DX microprocessor instructions operate on either 0, 1, 2 or 3 operands; where an operand resides in a register, in the instruction itself or in memory. Most zero operand instructions (e.g., CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2 bytes long. Since the Intel486 DX microprocessor has a 32-byte instruction queue, an average of 10 instructions will be prefetched. The use of two operands permits the following types of common instructions:

Register to Register
Memory to Register
Memory to Memory
Immediate to Register
Register to Memory
Immediate to Memory

The operands can be either 8, 16, or 32 bits long. As a general rule, when executing code written for the Intel486 DX, Intel486 or Intel386 microprocessors (32-bit code), operands are 8 or 32 bits; when executing existing 80286 or 8086 code (16-bit code), operands are 8 or 16 bits. Prefixes can be added to all instructions which override the default length of the operands (i.e., use 32-bit operands for 16-bit code, or 16-bit operands for 32-bit code).

2.3 Memory Organization

Introduction

Memory on the Intel486 DX microprocessor is divided up into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address, the high order byte at the high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address, the high-order byte at the highest address. The address of a word or dword is the byte address of the low-order byte.

In addition to these basic data types, the Intel486 DX microprocessor supports two larger units of memory: pages and segments. Memory can be divided up into one or more variable length segments, which can be swapped to disk or shared between programs. Memory can also be organized into one or more 4 Kbyte pages. Finally, both segmentation and paging can be combined, gaining the advantages of both systems. The Intel486 DX microprocessor supports both pages and segments in order to provide maximum flexibility to the system designer. Segmentation and paging are complementary. Segmentation is useful for organizing memory in logical modules, and as such is a tool for the application programmer, while pages are useful for the system programmer for managing the physical memory of a system.

2.3.1 ADDRESS SPACES

The Intel486 DX microprocessor has three distinct address spaces: **logical, linear,** and **physical**. A **logical** address (also known as a **virtual** address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, DISPLACEMENT) discussed in Section 2.5.3 **Memory Addressing Modes** into an effective address. Since each task on the Intel486 DX microprocessor has a maximum of 16K ($2^{14}-1$) selectors, and offsets can be 4 gigabytes, (2^{32} bits) this gives a total of 2^{46} bits or 64 terabytes of **logical** address space per task. The programmer sees this virtual address space.

The segmentation unit translates the **logical** address space into a 32-bit **linear** address space. If the paging unit is not enabled then the 32-bit **linear** address corresponds to the **physical** address. The paging unit translates the **linear** address space into the **physical** address space. The **physical address** is what appears on the address pins.

The primary difference between Real Mode and Protected Mode is how the segmentation unit performs the translation of the **logical** address into the **linear** address. In Real Mode, the segmentation unit shifts the selector left four bits and adds the result to the offset to form the **linear** address. While in Protected Mode every selector has a **linear** base address associated with it. The **linear base** address is stored in one of two operating system tables (i.e., the Local Descriptor Table or Global Descriptor Table). The selector's **linear base** address is added to the offset to form the final **linear** address.

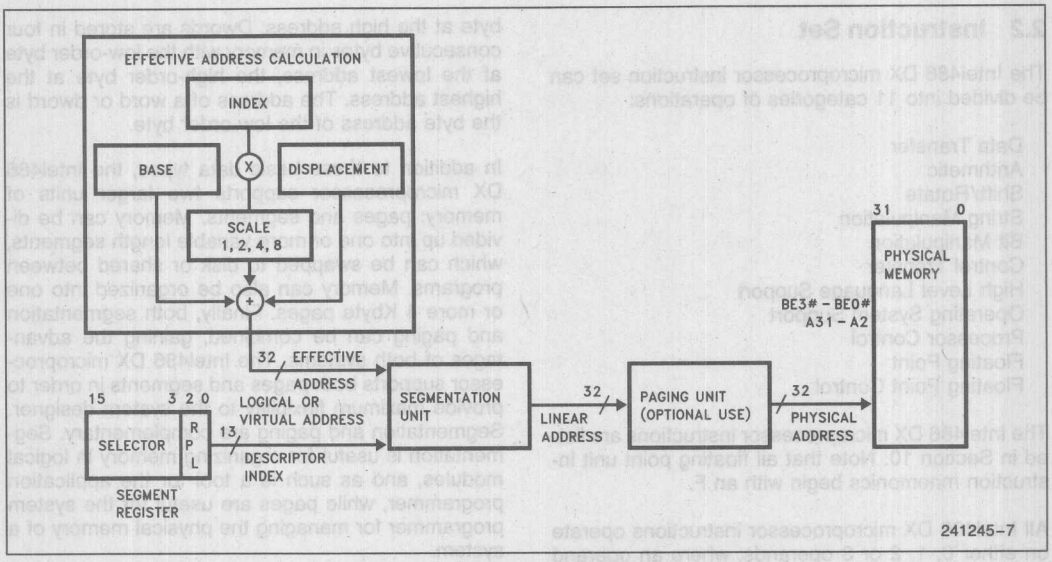


Figure 2.16. Address Translation

Figure 2.16 shows the relationship between the various address spaces.

2.3.2 SEGMENT REGISTER USAGE

The main data structure used to organize memory is the segment. On the Intel486 DX microprocessor, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments: code and data, the segments are of variable size and can be as small as 1 byte or as large as 4 gigabytes (2^{32} bytes).

In order to provide compact instruction encoding, and increase processor performance, instructions do not need to explicitly specify which segment register is used. A default segment register is automatically chosen according to the rules of Table 2.11 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register; Stack references use the SS register and Instruction fetches use the CS register. The contents of the Instruction Pointer provide the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 2.11. The override prefixes also allow the use of the ES, FS and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero and create a system with a four gigabyte linear address space. This creates a system where the virtual address space is the same as the linear address space. Further details of segmentation are discussed in Section 4.1.

2.4 I/O Space

The Intel486 DX microprocessor has two distinct physical address spaces: Memory and I/O. Generally, peripherals are placed in I/O space although the Intel486 DX microprocessor also supports memory-mapped peripherals. The I/O space consists of 64 Kbytes, it can be divided into 64K 8-bit ports, 32K 16-bit ports, or 16K 32-bit ports, or any combination of ports which add up to less than 64 Kbytes. The 64K I/O address space refers to physical memory rather than linear address since I/O instructions do not go through the segmentation or paging hardware. The M/IO# pin acts as an additional address line thus allowing the system designer to easily determine which address space the processor is accessing.

The I/O ports are accessed via the IN and OUT I/O instructions, with the port address supplied as an immediate 8-bit constant in the instruction or in the DX register. All 8- and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/IO# pin to be driven low.

I/O port addresses 00F8H through 00FFH are reserved for use by Intel.

Table 2.11. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment Use	Segment Override Prefixes Possible
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA Instructions	SS	None
Source of POP, POPA, POPF, IRET, RET instructions	SS	None
Destination of STOS, MOVS, REP STOS, REP MOVS Instructions (DI is Base Register)	ES	None
Other Data References, with Effective Address Using Base Register of:		
[EAX]	DS	All
[EBX]	DS	
[ECX]	DS	
[EDX]	DS	
[ESI]	DS	
[EDI]	DS	
[EBP]	SS	
[ESP]	SS	

2.5 Addressing Modes

2.5.1 ADDRESSING MODES OVERVIEW

The Intel486 DX microprocessor provides a total of 11 addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

2.5.2 REGISTER AND IMMEDIATE MODES

Two of the addressing modes provide for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8-, 16- or 32-bit general registers.

Immediate Operand Mode: The operand is included in the instruction as part of the opcode.

2.5.3 32-BIT MEMORY ADDRESSING MODES

The remaining 9 modes provide a mechanism for specifying the effective address of an operand. The linear address consists of two components: the segment base address and an effective address. The effective address is calculated by using combinations of the following four address elements:

DISPLACEMENT: An 8-, or 32-bit immediate value, following the instruction.

BASE: The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area.

INDEX: The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array, or a string of characters.

SCALE: The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. Scaled index

mode is especially useful for accessing arrays or structures.

Combinations of these 4 components make up the 9 additional addressing modes. There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions. The one exception is the simultaneous use of Base and Index components which requires one additional clock.

As shown in Figure 2.17, the effective address (EA) of an operand is calculated according to the following formula.

EA = Base Reg + (Index Reg * Scaling) + Displacement

Direct Mode: The operand's offset is contained as part of the instruction as an 8-, 16- or 32-bit displacement.
EXAMPLE: INC Word PTR [500]

Register Indirect Mode: A BASE register contains the address of the operand.
EXAMPLE: MOV [ECX], EDX

Based Mode: A BASE register's contents is added to a DISPLACEMENT to form the operand's offset.
EXAMPLE: MOV ECX, [EAX + 24]

Index Mode: An INDEX register's contents is added to a DISPLACEMENT to form the operand's offset.
EXAMPLE: ADD EAX, TABLE[ESI]

Scaled Index Mode: An INDEX register's contents is multiplied by a scaling factor which is added to a DISPLACEMENT to form the operand's offset.
EXAMPLE: IMUL EBX, TABLE[ESI*4],7

Based Index Mode: The contents of a BASE register is added to the contents of an INDEX register to form the effective address of an operand.
EXAMPLE: MOV EAX, [ESI] [EBX]

Based Scaled Index Mode: The contents of an INDEX register is multiplied by a SCALING factor and the result is added to the contents of a BASE register to obtain the operand's offset.
EXAMPLE: MOV ECX, [EDX*8] [EAX]

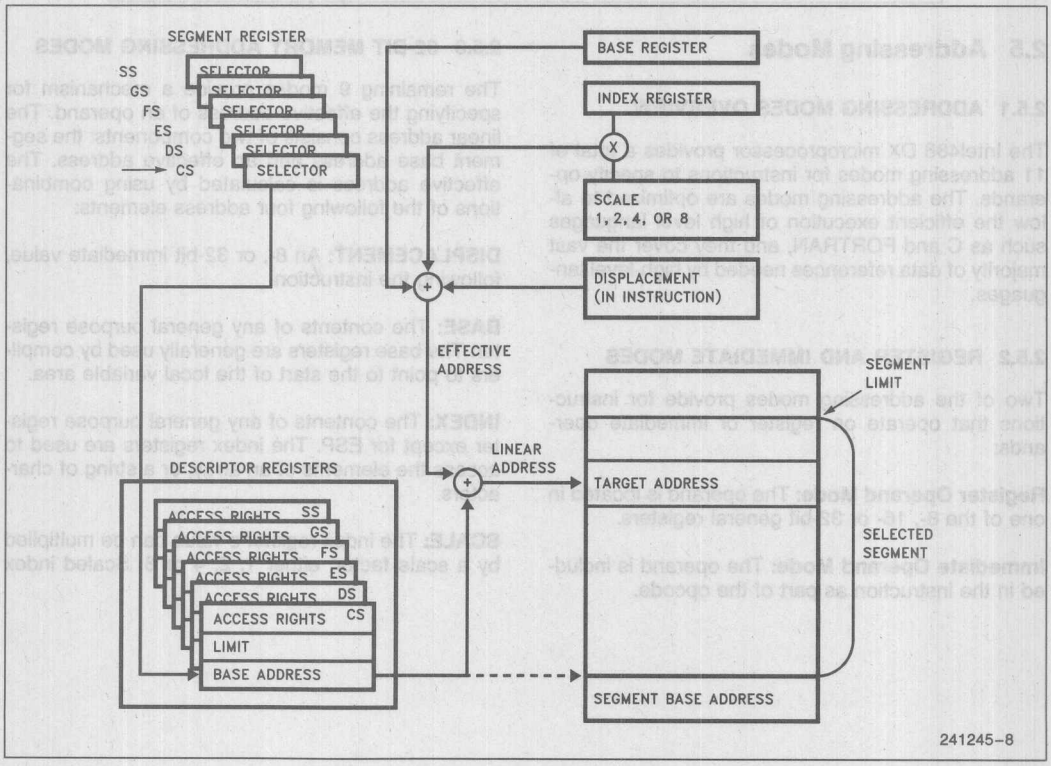


Figure 2.17. Addressing Mode Calculations

Based Index Mode with Displacement: The contents of an INDEX Register and a BASE register's contents and a DISPLACEMENT are all summed together to form the operand offset.
EXAMPLE: ADD EDX, [ESI] [EBP+00FFFFF0H]

Based Scaled Index Mode with Displacement: The contents of an INDEX register are multiplied by a SCALING factor, the result is added to the contents of a BASE register and a DISPLACEMENT to form the operand's offset.
EXAMPLE: MOV EAX, LOCALTABLE[EDI*4] [EBP+80]

2.5.4 DIFFERENCES BETWEEN 16- AND 32-BIT ADDRESSES

In order to provide software compatibility with the 80286 and the 8086, the Intel486 DX Microprocessor can execute 16-bit instructions in Real and Protected Modes. The processor determines the size of the instructions it is executing by examining the D bit in the CS segment Descriptor. If the D bit is 0 then all operand lengths and effective addresses are assumed to be 16 bits long. If the D bit is 1 then the default length for operands and addresses is 32 bits. In Real Mode the default size for operands and addresses is 16 bits.

Regardless of the default precision of the operands or addresses, the Intel486 DX Microprocessor is able to execute either 16- or 32-bit instructions. This is specified via the use of override prefixes. Two prefixes, the **Operand Size Prefix** and the **Address Length Prefix**, override the value of the D bit on an individual instruction basis. These prefixes are automatically added by Intel assemblers.

Example: The processor is executing in Real Mode and the programmer needs to access the EAX registers. The assembler code for this might be MOV EAX, 32-bit MEMORYOP, ASM486 Macro Assembler automatically determines that an Operand Size Prefix is needed and generates it.

Example: The D bit is 0, and the programmer wishes to use Scaled Index addressing mode to access an array. The Address Length Prefix allows the use of MOV DX, TABLE[ESI*2]. The assembler uses an Address Length Prefix since, with D=0, the default addressing mode is 16 bits.

Example: The D bit is 1, and the program wants to store a 16-bit quantity. The Operand Length Prefix is used to specify only a 16-bit value; MOV MEM16, DX.

The OPERAND LENGTH and Address Length Prefixes can be applied separately or in combination to any instruction. The Address Length Prefix does not allow addresses over 64 Kbytes to be accessed in Real Mode. A memory address which exceeds FFFFH will result in a General Protection Fault. An Address Length Prefix only allows the use of the additional Intel486 DX Microprocessor addressing modes.

When executing 32-bit code, the Intel486 DX Microprocessor uses either 8-, or 32-bit displacements, and any register can be used as base or index registers. When executing 16-bit code, the displacements are either 8, or 16 bits, and the base and index register conform to the 80286 model. Table 2.12 illustrates the differences.

2.6 Data Formats

2.6.1 DATA TYPES

The Intel486 DX Microprocessor can support a wide variety of data types. In the following descriptions, the on-chip floating point unit (FPU) consists of the floating point registers. The central processing unit (CPU) consists of the base architecture registers.

2.6.1.1 Unsigned Data Types

The FPU does not support unsigned data types. Refer to Table 2.13.

Byte: Unsigned 8-bit quantity

Word: Unsigned 16-bit quantity

Dword: Unsigned 32-bit quantity

The least significant bit (LSB) in a byte is bit 0, and the most significant bit is 7.

Table 2.12. BASE and INDEX Registers for 16- and 32-Bit Addresses

	16-Bit Addressing	32-Bit Addressing
BASE REGISTER	BX,BP	Any 32-bit GP Register
INDEX REGISTER	SI,DI	Any 32-bit GP Register Except ESP
SCALE FACTOR	none	1, 2, 4, 8
DISPLACEMENT	0, 8, 16 bits	0, 8, 32 bits

2.6.1.2 Signed Data Types

All signed data types assume 2's complement notation. The signed data types contain two fields, a sign bit and a magnitude. The sign bit is the most significant bit (MSB). The number is negative if the sign bit is 1. If the sign bit is 0, the number is positive. The magnitude field consists of the remaining bits in the number. Refer to Table 2.13.

8-bit Integer: Signed 8-bit quantity

16-bit Integer: Signed 16-bit quantity

32-bit Integer: Signed 32-bit quantity

64-bit Integer: Signed 64-bit quantity

The FPU only supports 16-, 32- and 64-bit integers. The CPU only supports 8-, 16- and 32-bit integers.

2.6.1.3 Floating Point Data Types

Floating point data type in the Intel486 DX microprocessor contain three fields, sign, significand and exponent. The sign field is one bit and is the MSB of the floating point number. The number is negative if the sign bit is 1. If the sign bit is 0, the number is positive. The significand gives the significant bits of the number. The exponent field contains the power of 2 needed to scale the significand. Refer to Table 2.13.

Only the FPU supports floating point data types.

Single Precision Real: 23-bit significand and 8-bit exponent. 32 bits total.

Double Precision Real: 52-bit significand and 11-bit exponent. 64 bits total.

Extended Precision Real: 64-bit significand and 15-bit exponent. 80 bits total.

2.6.1.4 BCD Data Types

The Intel486 DX microprocessor supports packed and unpacked binary coded decimal (BCD) data types. A packed BCD data type contains two digits per byte, the lower digit is in bits 0–3 and the upper digit in bits 4–7. An unpacked BCD data type contains 1 digit per byte stored in bits 0–3.

The CPU supports 8-bit packed and unpacked BCD data types. The FPU only supports 80-bit packed BCD data types. Refer to Table 2.13.

2.6.1.5 String Data Types

A string data type is a contiguous sequence of bits, bytes, words or dwords. A string may contain between 1 byte and 4 Gbytes. Refer to Table 2.14.

String data types are only supported by the CPU.

Byte String: Contiguous sequence of bytes.

Word String: Contiguous sequence of words.

Dword String: Contiguous sequence of dwords.

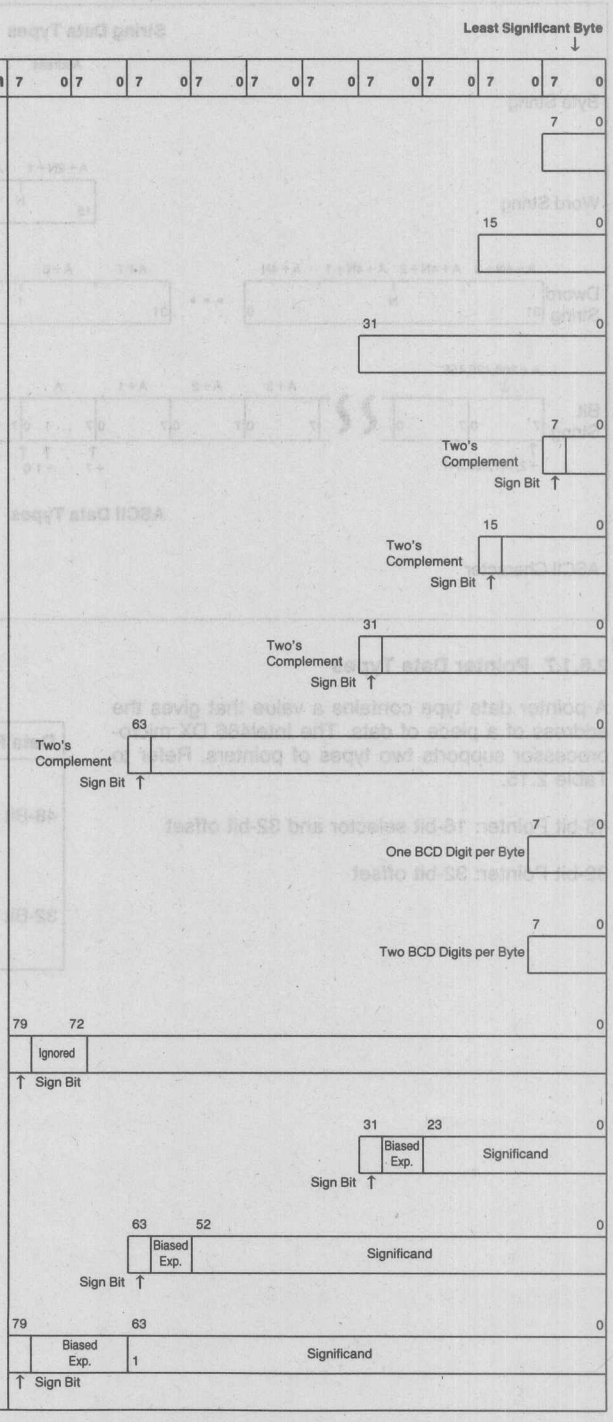
Bit String: A set of contiguous bits. In the Intel486 DX microprocessor bit strings can be up to 4 gigabits long.

2.6.1.6 ASCII Data Types

The Intel486 DX microprocessor supports ASCII (American Standard Code for Information Interchange) strings and can perform arithmetic operations (such as addition and division) on ASCII data. Refer to Table 2.14.

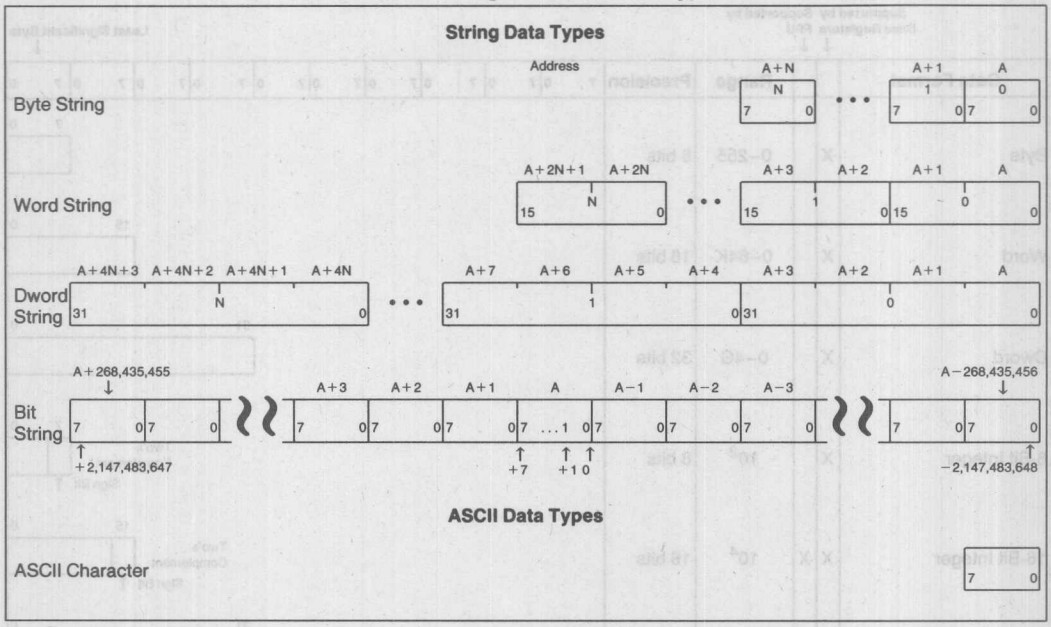
Table 2.13. Intel486™ DX Microprocessor Data Types

Data Format	Supported by Base Registers	Supported by FPU	Range	Precision
Byte	X		0–255	8 bits
Word	X		0–64K	16 bits
Dword	X		0–4G	32 bits
8-Bit Integer	X		10^2	8 bits
16-Bit Integer	X	X	10^4	16 bits
32-Bit Integer	X	X	10^9	32 bits
64-Bit Integer		X	10^{19}	64 bits
8-Bit Unpacked BCD	X		0–9	1 Digit
8-Bit Packed BCD	X		0–9	2 Digits
80-Bit Packed BCD		X	$\pm 10^{\pm 18}$	18 Digits
Single Precision Real		X	$\pm 10^{\pm 38}$	24 Bits
Double Precision Real		X	$\pm 10^{\pm 308}$	53 Bits
Extended Precision Real		X	$\pm 10^{\pm 4932}$	64 Bits



2

Table 2.14. String and ASCII Data Types



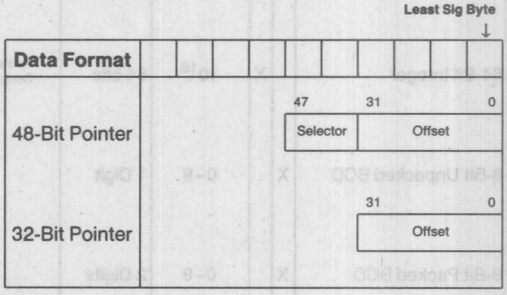
2.6.1.7 Pointer Data Types

A pointer data type contains a value that gives the address of a piece of data. The Intel486 DX microprocessor supports two types of pointers. Refer to Table 2.15.

48-bit Pointer: 16-bit selector and 32-bit offset

32-bit Pointer: 32-bit offset

Table 2.15. Pointer Data Types



2.6.2 LITTLE ENDIAN vs BIG ENDIAN DATA FORMATS

The Intel486 DX microprocessor, as well as all other members of the 86 architecture use the "little-endian" method for storing data types that are larger than one byte. Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address and the high order byte at the high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address and the high order byte at the highest address. The address of a word or dword data item is the byte address of the low-order byte.

Figure 2.18 illustrates the differences between the big-endian and little-endian formats for dwords. The 32 bits of data are shown with the low order bit numbered bit 0 and the high order bit numbered 32. Big-endian data is stored with the high-order bits at the lowest addressed byte. Little-endian data is stored with the high-order bits in the highest addressed byte.

The Intel486 DX microprocessor has two instructions which can convert 16- or 32-bit data between the two byte orderings. BSWAP (byte swap) handles four byte values and XCHG (exchange) handles two byte values.

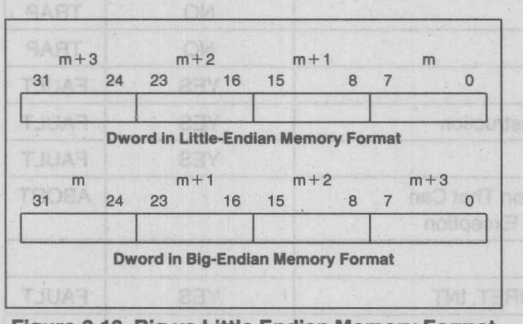


Figure 2.18. Big vs Little Endian Memory Format

2.7 Interrupts

2.7.1 INTERRUPTS AND EXCEPTIONS

Interrupts and exceptions alter the normal program flow, in order to handle external events, to report errors or exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT N instruction, the processor treats software interrupts as exceptions.

Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately **after** the interrupted instruction. Sections 2.7.3 and 2.7.4 discuss the differences between Maskable and Non-Maskable interrupts.

Exceptions are classified as faults, traps, or aborts depending on the way they are reported, and whether or not restart of the instruction causing the exception is supported. **Faults** are exceptions that are detected and serviced **before** the execution of the faulting instruction. A fault would occur in a virtual memory system, when the processor referenced a page or a segment which was not present. The operating system would fetch the page or segment from disk, and then the Intel486 DX microprocessor would restart the instruction. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. User defined interrupts are examples of traps. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined. Aborts are used to report severe errors, such as a hardware error, or illegal values in system tables.

Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand, the return address from an exception fault routine will always point at the instruction causing the exception and include any leading instruction prefixes. Table 2.16 summarizes the possible interrupts for the Intel486 DX microprocessor and shows where the return address points.

The Intel486 DX microprocessor has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. In Real Mode (see Section 3.1), the vectors are 4 byte quantities, a Code Segment plus a 16-bit offset; in Protected Mode, the interrupt vectors are 8 byte quantities, which are put in an Interrupt Descriptor Table (see Section 4.3.3.4). Of the 256 possible interrupts, 32 are reserved for use by Intel, the remaining 224 are free to be used by the system designer.

2.7.2 INTERRUPT PROCESSING

When an interrupt occurs the following actions happen. First, the current program address and the Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is sup-

plied to the Intel486 DX microprocessor which identifies the appropriate entry in the interrupt table. The table contains the starting address of the interrupt service routine. Then, the user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the Intel486 DX microprocessor in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

2.7.3 MASKABLE INTERRUPT

Maskable interrupts are the most common way used by the Intel486 DX microprocessor to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled high and the Interrupt Flag bit (IF) is enabled. The processor only responds to interrupts between instructions, (REPeat String instructions, have an "interrupt window", between memory moves, which allows interrupts during long string moves). When an interrupt occurs the processor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt, (one of 224 user defined interrupts). The exact nature of the interrupt sequence is discussed in Section 7.2.10.

Table 2.16. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	YES	FAULT
Debug Exception	1	Any Instruction	YES	TRAP*
NMI Interrupt	2	INT 2 or NMI	NO	NMI
One Byte Interrupt	3	INT	NO	TRAP
Interrupt on Overflow	4	INTO	NO	TRAP
Array Bounds Check	5	BOUND	YES	FAULT
Invalid OP-Code	6	Any Illegal Instruction	YES	FAULT
Device Not Available	7	ESC, WAIT	YES	FAULT
Double Fault	8	Any Instruction That Can Generate an Exception		ABORT
Intel Reserved	9			
Invalid TSS	10	JMP, CALL, IRET, INT	YES	FAULT
Segment Not Present	11	Segment Register Instructions	YES	FAULT
Stack Fault	12	Stack References	YES	FAULT
General Protection Fault	13	Any Memory Reference	YES	FAULT
Page Fault	14	Any Memory Access or Code Fetch	YES	FAULT
Intel Reserved	15			
Floating Point Error	16	Floating Point, WAIT	YES	FAULT
Alignment Check Interrupt	17	Unaligned Memory Access	YES	FAULT
Intel Reserved	18–31			
Two Byte Interrupt	0–255	INT n	NO	TRAP

*Some debug exceptions may report both traps on the previous instruction, and faults on the next instruction.

The IF bit in the EFLAG registers is reset when an interrupt is being serviced. This effectively disables servicing additional interrupts during an interrupt service routine. However, the IF may be set explicitly by the interrupt handler, to allow the nesting of interrupts. When an IRET instruction is executed the original state of the IF is restored.

2.7.4 NON-MASKABLE INTERRUPT

Non-maskable interrupts provide a method of servicing very high priority interrupts. A common example of the use of a non-maskable interrupt (NMI) would be to activate a power failure routine. When the NMI input is pulled high it causes an interrupt with an internally supplied vector value of 2. Unlike a normal hardware interrupt, no interrupt acknowledgment sequence is performed for an NMI.

While executing the NMI servicing procedure, the Intel486 DX microprocessor will not service further NMI requests until an interrupt return (IRET) instruction is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The IF bit is cleared at the beginning of an NMI interrupt to inhibit further INTR interrupts.

2.7.5 SOFTWARE INTERRUPTS

A third type of interrupt/exception for the Intel486 DX microprocessor is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table.

A special case of the two byte software interrupt INT n is the one byte INT 3, or breakpoint interrupt. By inserting this one byte instruction in a program, the user can set breakpoints in his program as a debugging tool.

A final type of software interrupt is the single step interrupt. It is discussed in Section 9.2.

2.7.6 INTERRUPT AND EXCEPTION PRIORITIES

Interrupts are externally-generated events. Maskable Interrupts (on the INTR input) and Non-Maskable Interrupts (on the NMI input) are recognized at instruction boundaries. When NMI and maskable INTR are **both** recognized at the **same** instruction boundary, the Intel486 DX microprocessor invokes the NMI service routine first. If, after the NMI service routine has been invoked, maskable interrupts are still enabled, then the Intel486 DX microprocessor will invoke the appropriate interrupt service routine.

Table 2.17a. Intel486™ DX Microprocessor Priority for Invoking Service Routines in Case of Simultaneous External Interrupts

1. NMI
2. INTR

Exceptions are internally-generated events. Exceptions are detected by the Intel486 DX microprocessor if, in the course of executing an instruction, the Intel486 DX microprocessor detects a problematic condition. The Intel486 DX microprocessor then immediately invokes the appropriate exception service routine. The state of the Intel486 DX microprocessor is such that the instruction causing the exception can be restarted. If the exception service routine has taken care of the problematic condition, the instruction will execute without causing the same exception.

It is possible for a single instruction to generate several exceptions (for example, transferring a single operand could generate two page faults if the operand location spans two "not present" pages). However, only one exception is generated upon each attempt to execute the instruction. Each exception service routine should correct its corresponding exception, and restart the instruction. In this manner, exceptions are serviced until the instruction executes successfully.

As the Intel486 DX microprocessor executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2.17b. This cycle is repeated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

Table 2.17b. Sequence of Exception Checking

Consider the case of the Intel486 DX microprocessor having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed:

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
3. Check for external NMI and INTR.
4. Check for Segmentation Faults that prevented fetching the entire next instruction (exceptions 11 or 13).
5. Check for Page Faults that prevented fetching the entire next instruction (exception 14).
6. Check for Faults decoding the next instruction (exception 6 if illegal opcode; exception 6 if in Real Mode or in Virtual 8086 Mode and attempting to execute an instruction for Protected Mode only (see Section 4.6.4); or exception 13 if instruction is longer than 15 bytes, or privilege violation in Protected Mode (i.e., not at IOPL or at CPL=0).
7. If WAIT opcode, check if TS=1 and MP=1 (exception 7 if both are 1).
8. If opcode for Floating Point Unit, check if EM=1 or TS=1 (exception 7 if either are 1).
9. If opcode for Floating Point Unit (FPU), check FPU error status (exception 16 if error status is asserted).
10. Check in the following order for each memory reference required by the instruction:
    a. Check for Segmentation Faults that prevent transferring the entire memory quantity (exceptions 11, 12, 13).
    b. Check for Page Faults that prevent transferring the entire memory quantity (exception 14).

NOTE:
The order stated supports the concept of the paging mechanism being "underneath" the segmentation mechanism. Therefore, for any given code or data reference in memory, segmentation exceptions are generated before paging exceptions are generated.

2.7.7 INSTRUCTION RESTART

The Intel486 DX microprocessor fully supports restarting all instructions after faults. If an exception is detected in the instruction to be executed (exception categories 4 through 10 in Table 2.17b), the Intel486 DX microprocessor invokes the appropriate exception service routine. The Intel486 DX microprocessor is in a state that permits restart of the instruction, for all cases but those in Table 2.17c. Note that all such cases are easily avoided by proper design of the operating system.

Table 2.17c. Conditions Preventing Instruction Restart

An instruction causes a task switch to a task whose Task State Segment is **partially** "not present". (An entirely "not present" TSS is restartable.) Partially present TSS's can be avoided either by keeping the TSS's of such tasks present in memory, or by aligning TSS segments to reside entirely within a single 4K page (for TSS segments of 4 Kbytes or less).

NOTE:
These conditions are avoided by using the operating system designs mentioned in this table.

2.7.8 DOUBLE FAULT

A Double Fault (exception 8) results when the processor attempts to invoke an exception service routine for the segment exceptions (10, 11, 12 or 13), but in the process of doing so, detects an exception other than a Page Fault (exception 14).

A Double Fault (exception 8) will also be generated when the processor attempts to invoke the Page Fault (exception 14) service routine, and detects an exception other than a second Page Fault. In any functional system, the entire Page Fault service routine must remain "present" in memory.

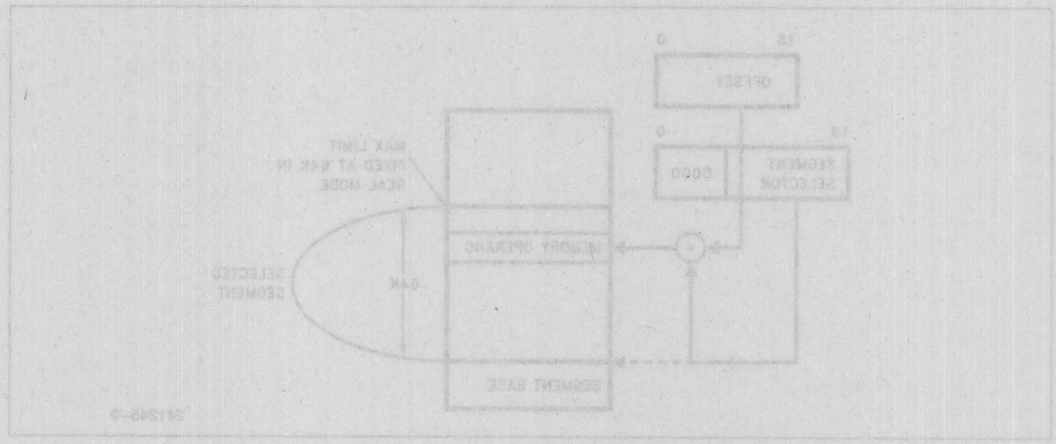
When a Double Fault occurs, the Intel486 DX microprocessor invokes the exception service routine for exception 8.

2.7.9 FLOATING POINT INTERRUPT VECTORS

Several interrupt vectors of the Intel486 DX microprocessor are used to report exceptional conditions while executing numeric programs in either real or protected mode. Table 2.18 shows these interrupts and their causes.

Table 2.18. Interrupt Vectors Used by FPU

Interrupt Number	Cause of Interrupt
7	A Floating Point instruction was encountered when EM or TS of the Intel486 DX processor control register zero (CR0) was set. EM = 1 indicates that software emulation of the instruction is required. When TS is set, either a Floating Point or WAIT instruction causes interrupt 7. This indicates that the current FPU context may not belong to the current task.
13	The first word or doubleword of a numeric operand is not entirely within the limit of its segment. The return address pushed onto the stack of the exception handler points at the Floating Point instruction that caused the exception, including any prefixes. The FPU has not executed this instruction; the instruction pointer and data pointer register refer to a previous, correctly executed instruction.
16	The previous numerics instruction caused an unmasked exception. The address of the faulty instruction and the address of its operand are stored in the instruction pointer and data pointer registers. Only Floating Point and WAIT instructions can cause this interrupt. The Intel486™ DX processor return address pushed onto the stack of the exception handler points to a WAIT or Floating Point instruction (including prefixes). This instruction can be restarted after clearing the exception condition in the FPU. The FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE instructions cannot cause this interrupt.



3.0 REAL MODE ARCHITECTURE

3.1 Real Mode Introduction

When the processor is reset or powered up it is initialized in Real Mode. Real Mode has the same base architecture as the 8086, but allows access to the 32-bit register set of the Intel486 microprocessor family. The addressing mechanism, memory size, interrupt handling, are all identical to the Real Mode on the 80286.

All of the Intel486 microprocessor instructions are available in Real Mode (except those instructions listed in Section 4.6.4). The default operand size in Real Mode is 16 bits, just like the 8086. In order to use the 32-bit registers and addressing modes, override prefixes must be used. In addition, the segment size on the Intel486 microprocessor in Real Mode is 64 Kbytes so 32-bit effective addresses must have a value less the 0000FFFFH. The primary purpose of Real Mode is to set up the processor for Protected Mode Operation.

The LOCK prefix on the Intel486 microprocessor, even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the Intel486 microprocessor in Protected Mode and Virtual 8086 Mode. Paging makes it impossible to guarantee that repeated string instructions can be LOCKed. The Intel486 microprocessor can't require that all pages holding the string be physically present in memory. Hence, a Page Fault (exception 14) might have to be taken during the repeated string instruction. Therefore the LOCK prefix can't be supported during repeated string instructions.

These are the only instruction forms where the LOCK prefix is legal on the Intel486 DX2 microprocessor:

Opcode	Operands (Dest, Source)
BIT Test and SET/RESET/COMPLEMENT	Mem, Reg/immed
XCHG	Reg, Mem
XCHG	Mem, Reg
ADD, OR, ADC, SBB, AND, SUB, XOR	Mem, Reg/immed
NOT, NEG, INC, DEC	Mem
CMPXCHG, XADD	Mem, Reg

An exception 6 will be generated if a LOCK prefix is placed before any instruction form or opcode not listed above. The LOCK prefix allows indivisible read/modify/write operations on memory operands using the instructions above. For example, even the ADD Reg, Mem is not LOCKable, because the Mem operand is not the destination (and therefore no memory read/modify/operation is being performed).

Since, on the Intel486 microprocessor, repeated string instructions are not LOCKable, it is not possible to LOCK the bus for a long period of time. Therefore, the LOCK prefix is not IOPL-sensitive on the Intel486 microprocessor. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed above.

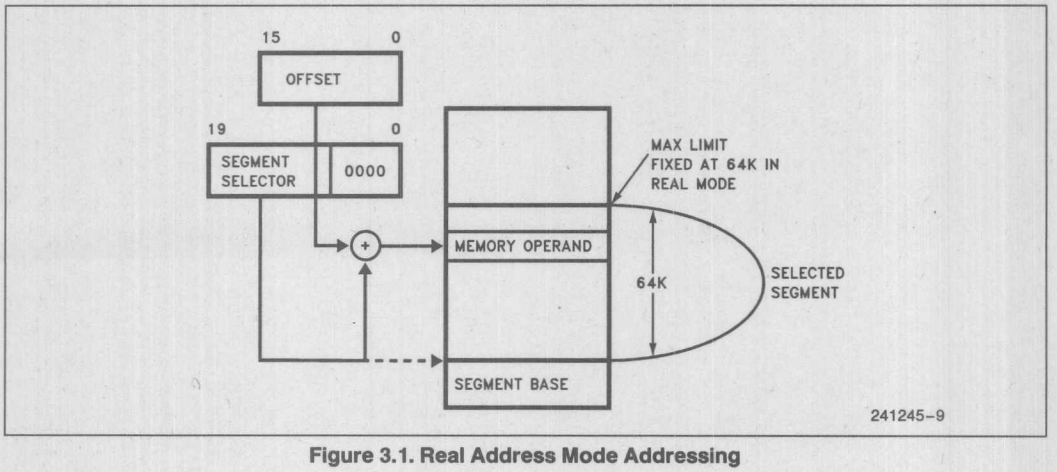


Figure 3.1. Real Address Mode Addressing

3.2 Memory Addressing

In Real Mode the maximum memory size is limited to 1 megabyte. Thus, only address lines A2–A19 are active. (Exception, after RESET address lines A20–A31 are high during CS-relative memory cycles until an intersegment jump or call is executed (see Section 6.5)).

Since paging is not allowed in Real Mode the linear addresses are the same as physical addresses. Physical addresses are formed in Real Mode by adding the contents of the appropriate segment register which is shifted left by four bits to an effective address. This addition results in a physical address from 00000000H to 0010FFEFH. This is compatible with 80286 Real Mode. Since segment registers are shifted left by 4 bits, Real Mode segments always start on 16 byte boundaries.

All segments in Real Mode are exactly 64 Kbytes long, and may be read, written, or executed. The Intel486 microprocessor will generate an exception 13 if a data operand or instruction fetch occurs past the end of a segment (i.e., if an operand has an offset greater than FFFFH, for example a word with a low byte at FFFFH and the high byte at 0000H).

Segments may be overlapped in Real Mode. Thus, if a particular segment does not use all 64 Kbytes another segment can be overlayed on top of the unused portion of the previous segment. This allows the programmer to minimize the amount of physical memory needed for a program.

3.3 Reserved Locations

There are two fixed areas in memory which are reserved in Real address mode: system initialization area and the interrupt table area. Locations 00000H through 003FFH are reserved for interrupt vectors. Each one of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations FFFFFFF0H through FFFFFFFFH are reserved for system initialization.

3.4 Interrupts

Many of the exceptions shown in Table 2.16 and discussed in Section 2.7 are not applicable to Real Mode operation, in particular exceptions 10, 11, 14, 17, will not happen in Real Mode. Other exceptions have slightly different meanings in Real Mode; Table 3.1 identifies these exceptions.

3.5 Shutdown and Halt

The HLT instruction stops program execution and prevents the processor from using the local bus until restarted. Either NMI, INTR with interrupts enabled (IF = 1), or RESET will force the Intel486 DX2 microprocessor out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

As in the case in protected mode, the shutdown will occur when a severe error is detected that prevents further processing. In Real Mode, shutdown can occur under two conditions:

An interrupt or an exception occur (exceptions 8 or 13) and the interrupt vector is larger than the Interrupt Descriptor Table (i.e., there is not an interrupt handler for the interrupt).

A CALL, INT or PUSH instruction attempts to wrap around the stack segment when SP is not even (i.e., pushing a value on the stack when SP = 0001 resulting in a stack segment greater than FFFFH).

An NMI input can bring the processor out of shutdown if the Interrupt Descriptor Table limit is large enough to contain the NMI interrupt vector (at least 0017H) and the stack has enough room to contain the vector and flag information (i.e., SP is greater than 0005H). If these conditions are not met, the Intel486 DX2 CPU is unable to execute the NMI and executes another shutdown cycle. In this case, the processor remains in the shutdown and can only exit via the RESET input.

Table 3.1. Exceptions with Different Meanings in Real Mode (see Table 2.16)

Function	Interrupt Number	Related Instructions	Return Address Location
Interrupt table limit too small	8	INT Vector is not within table limit	Before Instruction
CS, DS, ES, FS, GS Segment overrun exception	13	Word memory reference beyond offset = FFFFH. An attempt to execute past the end of CS segment.	Before Instruction
SS Segment overrun exception	12	Stack Reference beyond offset = FFFFH	Before Instruction

4.0 PROTECTED MODE ARCHITECTURE

4.1 Introduction

The complete capabilities of the Intel486 DX2 microprocessor are unlocked when the processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to four gigabytes (2^{32} bytes) and allows the running of virtual memory programs of almost unlimited size (64 terabytes or 2^{46} bytes). In addition Protected Mode allows the Intel486 DX2 microprocessor to run all of the existing 8086, 80286 and Intel386 microprocessor software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions especially optimized for supporting multitasking operating systems. The base architecture of the Intel486 microprocessor remains the same, the registers, instructions, and addressing modes described in the previous sections are retained. The main difference between Protected Mode, and Real Mode from a programmer's view is the increased address space, and a different addressing mechanism.

4.2 Addressing Mechanism

Like Real Mode, Protected Mode uses two components to form the logical address, a 16-bit selector is used to determine the linear base address of a segment, the base address is added to a 32-bit effective address to form a 32-bit linear address. The linear address is then either used as the 32-bit physical address, or if paging is enabled the paging mechanism maps the 32-bit linear address into a 32-bit physical address.

The difference between the two modes lies in calculating the base address. In Protected Mode the selector is used to specify an index into an operating system defined table (see Figure 4.1). The table contains the 32-bit base address of a given segment. The physical address is formed by adding the base address obtained from the table to the offset.

Paging provides an additional memory management mechanism which operates only in Protected Mode. Paging provides a means of managing the very large segments of the Intel486 microprocessor family. As such, paging operates beneath segmentation. The paging mechanism translates the protected linear address which comes from the segmentation unit into a physical address. Figure 4.2 shows the complete Intel486 DX2 addressing mechanism with paging enabled.

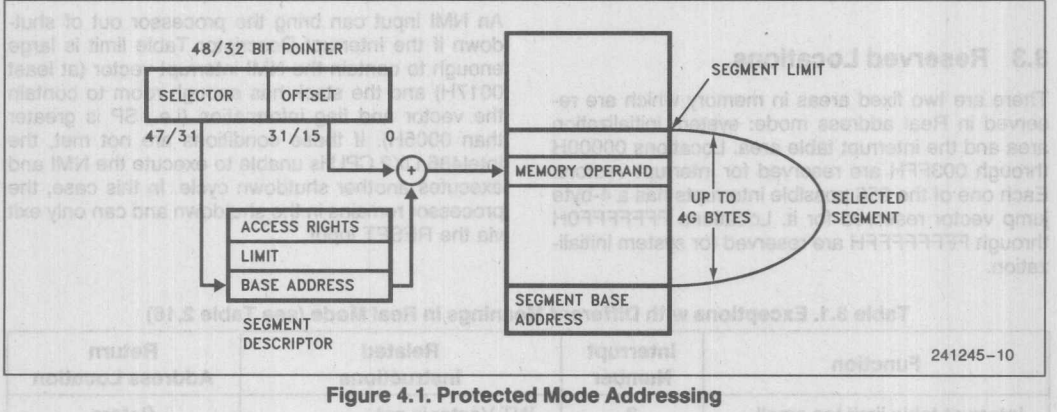


Figure 4.1. Protected Mode Addressing

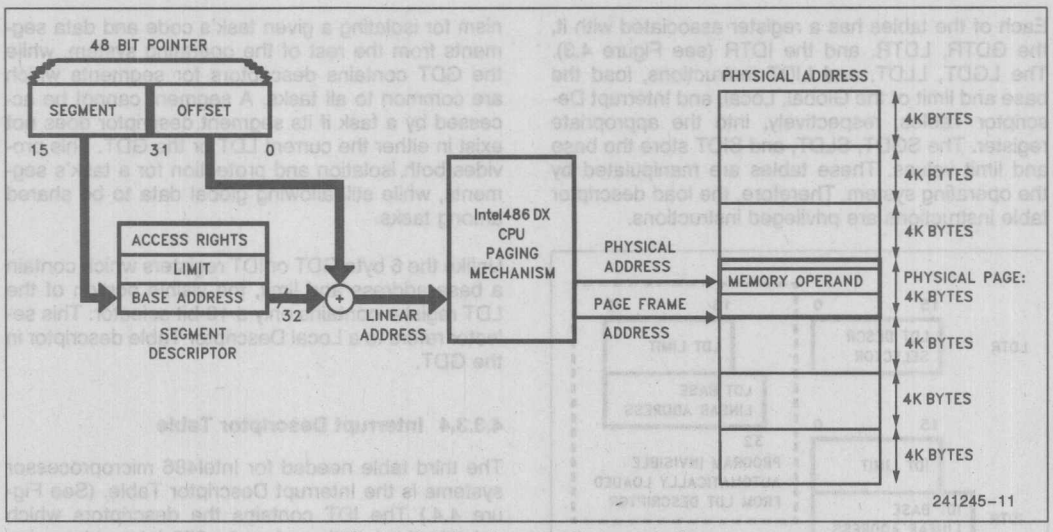


Figure 4.2. Paging and Segmentation

4.3 Segmentation

4.3.1 SEGMENTATION INTRODUCTION

Segmentation is one method of memory management. Segmentation provides the basis for protection. Segments are used to encapsulate regions of memory which have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about a segment is stored in an 8 byte data structure called a descriptor. All of the descriptors in a system are contained in tables recognized by hardware.

4.3.2 TERMINOLOGY

The following terms are used throughout the discussion of descriptors, privilege levels and protection:

PL: Privilege Level—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged. More privileged levels are numerically smaller than less privileged levels.

RPL: Requestor Privilege Level—The privilege level of the original supplier of the selector. RPL is determined by the **least two** significant bits of a selector.

DPL: Descriptor Privilege Level—This is the least privileged level at which a task may access that descriptor (and the segment associated with that descriptor). Descriptor Privilege Level is determined by bits 6:5 in the Access Right Byte of a descriptor.

CPL: Current Privilege Level—The privilege level at which a task is currently executing, which equals the privilege level of the code segment being executed. CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.

EPL: Effective Privilege Level—The effective privilege level is the least privileged of the RPL and DPL. Since smaller privilege level **values** indicate greater privilege, EPL is the numerical maximum of RPL and DPL.

Task: One instance of the execution of a program. Tasks are also referred to as processes.

4.3.3 DESCRIPTOR TABLES

4.3.3.1 Descriptor Tables Introduction

The descriptor tables define all of the segments which are used in a Intel486 microprocessor system. There are three types of tables on the Intel486 DX2 microprocessor which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays. They can range in size between 8 bytes and 64 Kbytes. Each table can hold up to 8192 8-byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address, and the 16-bit limit of each table.

Each of the tables has a register associated with it, the GDTR, LDTR, and the IDTR (see Figure 4.3). The LGDT, LLDT, and LIDT instructions, load the base and limit of the Global, Local, and Interrupt Descriptor Tables, respectively, into the appropriate register. The SGDT, SLDT, and SIDT store the base and limit values. These tables are manipulated by the operating system. Therefore, the load descriptor table instructions are privileged instructions.

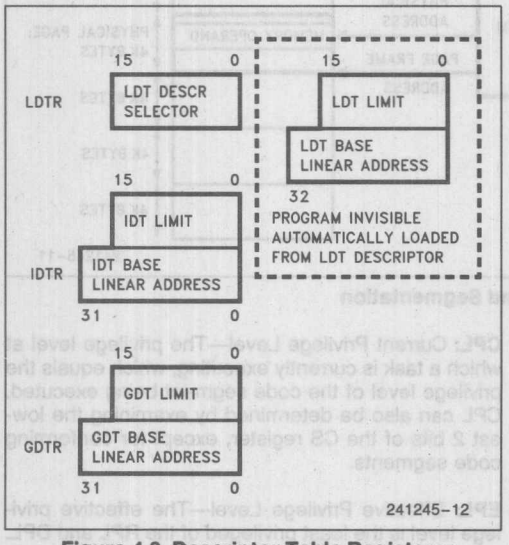


Figure 4.3. Descriptor Table Registers

4.3.3.2 Global Descriptor Table

The Global Descriptor Table (GDT) contains descriptors which are possibly available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for descriptors which are used for servicing interrupts (i.e., interrupt and trap descriptors). Every Intel486 microprocessor system contains a GDT. Generally the GDT contains code and data segments used by the operating systems and task state segments, and descriptors for the LDTs in a system.

The first slot of the Global Descriptor Table corresponds to the null selector and is not used. The null selector defines a null pointer value.

4.3.3.3 Local Descriptor Table

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks. A segment cannot be accessed by a task if its segment descriptor does not exist in either the current LDT or the GDT. This provides both isolation and protection for a task's segments, while still allowing global data to be shared among tasks.

Unlike the 6 byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT.

4.3.3.4 Interrupt Descriptor Table

The third table needed for Intel486 microprocessor systems is the Interrupt Descriptor Table. (See Figure 4.4.) The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced via INT instructions, external interrupt vectors, and exceptions. (See Section 2.7 **Interrupts).**

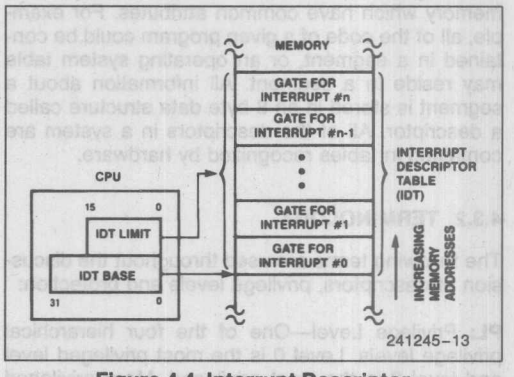


Figure 4.4. Interrupt Descriptor Table Register Use

4.3.4 DESCRIPTORS

4.3.4.1 Descriptor Attribute Bits

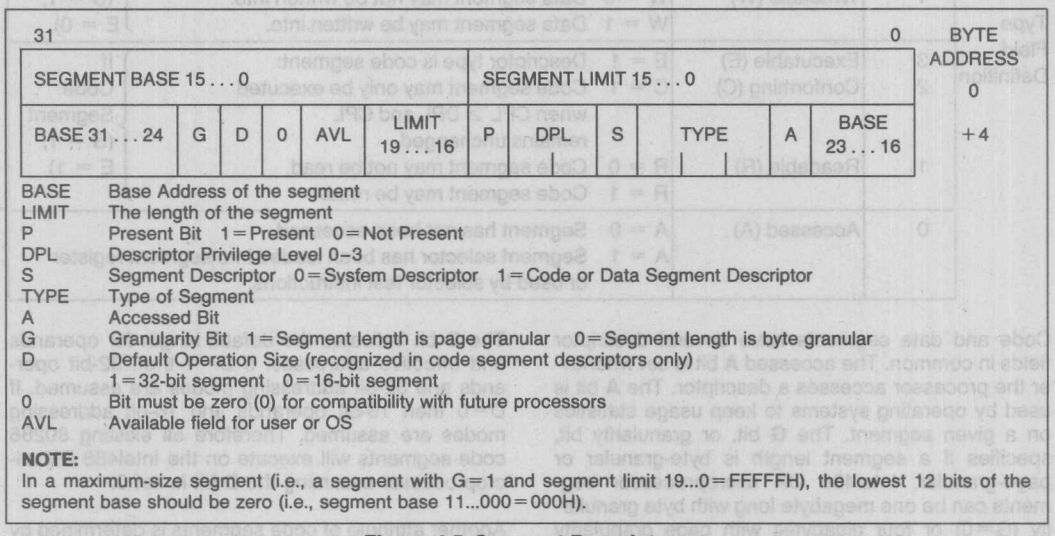
The object to which the segment selector points to is called a descriptor. Descriptors are eight byte quantities which contain attributes about a given region of linear address space (i.e., a segment). These

attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or 32-bit), and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 4.5 shows the general format of a descriptor. All segments on the Intel486 microprocessor have three attribute fields in common: the **P** bit, the **DPL** bit, and the **S** bit. The Present **P** bit is 1 if the segment is loaded in physical memory, if P=0 then any attempt to access this segment causes a not present exception (exception 11). The Descriptor Privilege Level **DPL** is a two-bit field which specifies the protection level 0–3 associated with a segment.

The Intel486 DX microprocessor has two main categories of segments: system segments and non-system segments (for code and data). The segment **S** bit in the segment descriptor determines if a given segment is a system segment or a code or data segment. If the S bit is 1 then the segment is either a code or data segment, if it is 0 then the segment is a system segment.

4.3.4.2 Intel486 CPU Code, Data Descriptors (S=1)

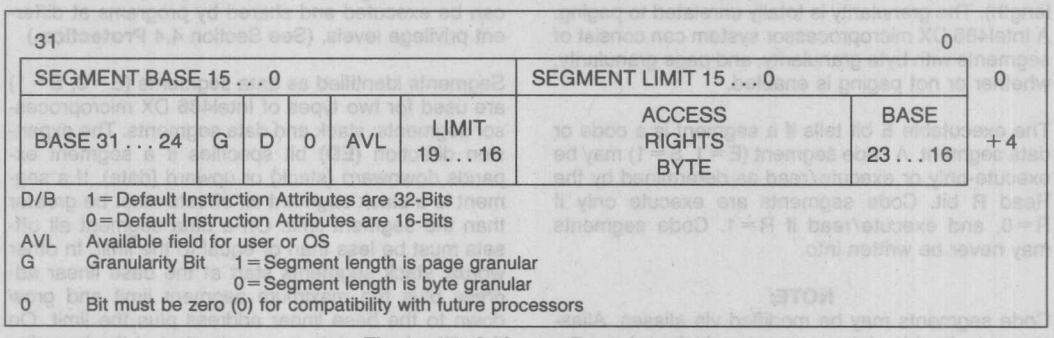
Figure 4.6 shows the general format of a code and data descriptor and Table 4.1 illustrates how the bits in the Access Rights Byte are interpreted.



BASE	Base Address of the segment
LIMIT	The length of the segment
P	Present Bit 1=Present 0=Not Present
DPL	Descriptor Privilege Level 0–3
S	Segment Descriptor 0=Sysfem Descriptor 1=Code or Data Segment Descriptor
TYPE	Type of Segment
A	Accessed Bit
G	Granularity Bit 1=Segment length is page granular 0=Segment length is byte granular
D	Default Operation Size (recognized in code segment descriptors only) 1=32-bit segment 0=16-bit segment
0	Bit must be zero (0) for compatibility with future processors
AVL	Available field for user or OS

NOTE:
In a maximum-size segment (i.e., a segment with G=1 and segment limit 19...0=FFFFFH), the lowest 12 bits of the segment base should be zero (i.e., segment base 11...000=000H).

Figure 4.5. Segment Descriptors



D/B	1=Default Instruction Attributes are 32-Bits 0=Default Instruction Attributes are 16-Bits
AVL	Available field for user or OS
G	Granularity Bit 1=Segment length is page granular 0=Segment length is byte granular
0	Bit must be zero (0) for compatibility with future processors

Figure 4.6. Segment Descriptors

Table 4.1. Access Rights Byte Definition for Code and Data Descriptions

	Bit Position	Name	Function		
	7	Present (P)	P = 1	Segment is mapped into physical memory.	
			P = 0	No mapping to physical memory exits, base and limit are not used.	
	6–5	Descriptor Privilege Level (DPL)		Segment privilege attribute used in privilege tests.	
	4	Segment Descriptor (S)	S = 1	Code or Data (includes stacks) segment descriptor.	
			S = 0	System Segment Descriptor or Gate Descriptor.	
Type Field Definition	3	Executable (E)	E = 0	Descriptor type is data segment:	If Data Segment (S = 1, E = 0)
	2	Expansion Direction (ED)	ED = 0	Expand up segment, offsets must be ≤ limit.	
			ED = 1	Expand down segment, offsets must be > limit.	
	1	Writeable (W)	W = 0	Data segment may not be written into.	
			W = 1	Data segment may be written into.	
	3	Executable (E)	E = 1	Descriptor type is code segment:	If Code Segment (S = 1, E = 1)
	2	Conforming (C)	C = 1	Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.	
	1	Readable (R)	R = 0	Code segment may not be read.	
			R = 1	Code segment may be read.	
	0	Accessed (A)	A = 0	Segment has not been accessed.	
			A = 1	Segment selector has been loaded into segment register or used by selector test instructions.	

Code and data segments have several descriptor fields in common. The accessed **A** bit is set whenever the processor accesses a descriptor. The **A** bit is used by operating systems to keep usage statistics on a given segment. The **G** bit, or granularity bit, specifies if a segment length is byte-granular or page-granular. Intel486 DX microprocessor segments can be one megabyte long with byte granularity (G=0) or four gigabytes with page granularity (G=1), (i.e., 2^{20} pages each page is 4 Kbytes in length). The granularity is totally unrelated to paging. A Intel486 DX microprocessor system can consist of segments with byte granularity, and page granularity, whether or not paging is enabled.

The executable **E** bit tells if a segment is a code or data segment. A code segment (E=1, S=1) may be execute-only or execute/read as determined by the Read **R** bit. Code segments are execute only if R=0, and execute/read if R=1. Code segments may never be written into.

NOTE:
Code segments may be modified via aliases. Aliases are writeable data segments which occupy the same range of linear address space as the code segment.

The **D** bit indicates the default length for operands and effective addresses. If D=1 then 32-bit operands and 32-bit addressing modes are assumed. If D=0 then 16-bit operands and 16-bit addressing modes are assumed. Therefore all existing 80286 code segments will execute on the Intel486 DX microprocessor assuming the D bit is set 0.

Another attribute of code segments is determined by the conforming **C** bit. Conforming segments, C=1, can be executed and shared by programs at different privilege levels. (See Section 4.4 **Protection**.)

Segments identified as data segments (E=0, S=1) are used for two types of Intel486 DX microprocessor segments: stack and data segments. The expansion direction **(ED)** bit specifies if a segment expands downward (stack) or upward (data). If a segment is a stack segment all offsets must be greater than the segment limit. On a data segment all offsets must be less than or equal to the limit. In other words, stack segments start at the base linear address plus the maximum segment limit and grow down to the base linear address plus the limit. On the other hand, data segments start at the base linear address and expand to the base linear address plus limit.

The write **W** bit controls the ability to write into a segment. Data segments are read-only if W = 0. The stack segment must have W = 1.

The **B** bit controls the size of the stack pointer register. If B = 1, then PUSHes, POPs, and CALLs all use the 32-bit ESP register for stack references and assume an upper limit of FFFFFFFFH. If B = 0, stack instructions all use the 16-bit SP register and assume an upper limit of FFFFH.

4.3.4.3 System Descriptor Formats

System segments describe information about operating system tables, tasks, and gates. Figure 4.7 shows the general format of system segment descriptors, and the various types of system segments. Intel486 DX microprocessor system descriptors contain a 32-bit base linear address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zero.

4.3.4.4 LDT Descriptors (S = 0, TYPE = 2)

LDT descriptors (S = 0, TYPE = 2) contain information about Local Descriptor Tables. LDTs contain a table of segment descriptors, unique to a particular task. Since the instruction to load the LDTR is only available at privilege level 0, the DPL field is ignored. LDT descriptors are only allowed in the Global Descriptor Table (GDT).

4.3.4.5 TSS Descriptors (S = 0, TYPE = 1, 3, 9, B)

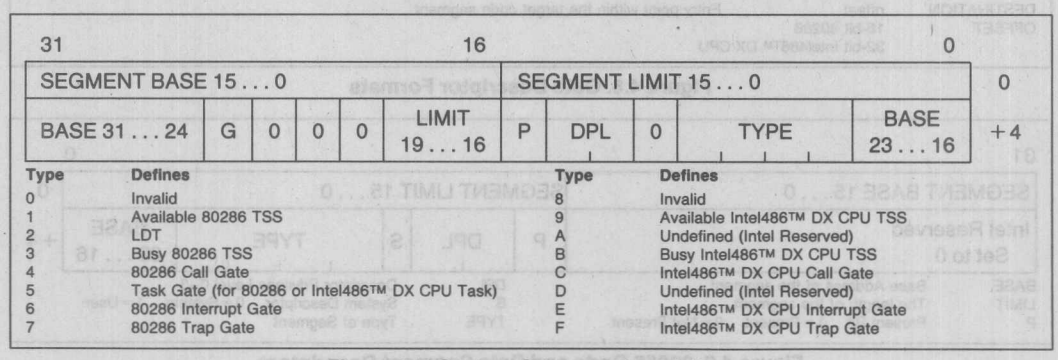
A Task State Segment (TSS) descriptor contains information about the location, size, and privilege level of a Task State Segment (TSS). A TSS in turn is a special fixed format segment which contains all the state information for a task and a linkage field to permit nesting tasks. The TYPE field is used to indicate whether the task is currently BUSY (i.e., on a chain of active tasks) or the TSS is available. The TYPE field also indicates if the segment contains a 80286 or an Intel486 DX microprocessor TSS. The Task Register (TR) contains the selector which points to the current Task State Segment.

4.3.4.6 Gate Descriptors (S = 0, TYPE = 4–7, C, F)

Gates are used to control access to entry points within the target code segment. The various types of gate descriptors are **call** gates, **task** gates, **interrupt** gates, and **trap** gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the processor to automatically perform protection checks. It also allows system designers to control entry points to the operating system. Call gates are used to change privilege levels (see Section 4.4 **Protection**), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines.

Figure 4.8 shows the format of the four types of gate descriptors. Call gates are primarily used to transfer program control to a more privileged level. The call gate descriptor consists of three fields: the access byte, a long pointer (selector and offset) which points to the start of a routine and a word count which specifies how many parameters are to be copied from the caller's stack to the stack of the called routine. The word count field is only used by call gates when there is a change in the privilege level, other types of gates ignore the word count field.

Interrupt and trap gates use the destination selector and destination offset fields of the gate descriptor as a pointer to the start of the interrupt or trap handler routines. The difference between interrupt gates and trap gates is that the interrupt gate disables interrupts (resets the IF bit) while the trap gate does not.



Type	Defines	Type	Defines
0	Invalid	8	Invalid
1	Available 80286 TSS	9	Available Intel486™ DX CPU TSS
2	LDT	A	Undefined (Intel Reserved)
3	Busy 80286 TSS	B	Busy Intel486™ DX CPU TSS
4	80286 Call Gate	C	Intel486™ DX CPU Call Gate
5	Task Gate (for 80286 or Intel486™ DX CPU Task)	D	Undefined (Intel Reserved)
6	80286 Interrupt Gate	E	Intel486™ DX CPU Interrupt Gate
7	80286 Trap Gate	F	Intel486™ DX CPU Trap Gate

Figure 4.7. System Segment Descriptors

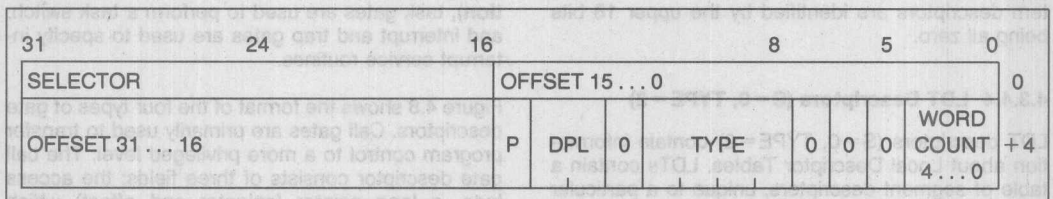
Task gates are used to switch tasks. Task gates may only refer to a task state segment (see Section 4.4.6 **Task Switching**) therefore only the destination selector portion of a task gate descriptor is used, and the destination offset is ignored.

Exception 13 is generated when a destination selector does not refer to a correct descriptor type, i.e., a code segment for an interrupt, trap or call gate, a TSS for a task gate.

The access byte format is the same for all gate descriptors. P=1 indicates that the gate contents are valid. P=0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (see Section 4.4 **Protection**). The S field, bit 4 of the access rights byte, must be 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 4.8.

4.3.4.7 Differences Between Intel486 DX2 Microprocessor and 80286 Descriptors

In order to provide operating system compatibility between the 80286 and Intel486 DX2 microprocessor, the Intel486 DX microprocessor supports all of the 80286 segment descriptors. Figure 4.9 shows the general format of an 80286 system segment descriptor. The only differences between 80286 and Intel486 DX microprocessor descriptor formats are that the values of the type fields, and the limit and base address fields have been expanded for the Intel486 DX microprocessors. The 80286 system segment descriptors contained a 24-bit base address and 16-bit limit, while the Intel486 DX microprocessor system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit.



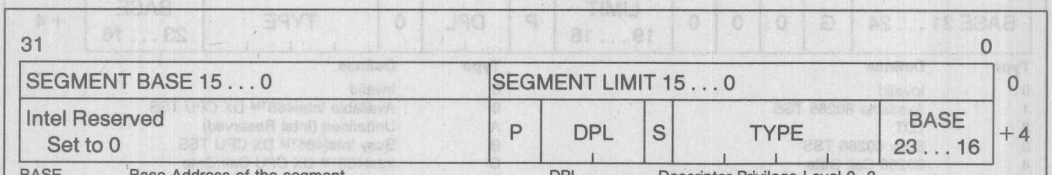
Gate Descriptor Fields

Name	Value	Description
Type	4	80286 call gate
	5	Task gate (for 80286 or Intel486™ DX CPU task)
	6	80286 interrupt gate
	7	80286 trap gate
	C	Intel486™ DX CPU call gate
	E	Intel486™ DX CPU interrupt gate
	F	Intel486™ DX CPU trap gate
P	0	Descriptor contents are not valid
	1	Descriptor contents are valid

DPL—least privileged level at which a task may access the gate. WORD COUNT 0–31—the number of parameters to copy from caller's stack to the called procedure's stack. The parameters are 32-bit quantities for Intel486™ DX CPU gates, and 16-bit quantities for 80286 gates.

DESTINATION SELECTOR	16-bit selector	Selector to the target code segment or Selector to the target task state segment for task gate
DESTINATION OFFSET	offset 16-bit 80286 32-bit Intel486™ DX CPU	Entry point within the target code segment

Figure 4.8. Gate Descriptor Formats



BASE	Base Address of the segment	DPL	Descriptor Privilege Level 0–3
LIMIT	The length of the segment	S	System Descriptor 0=System 1=User
P	Present Bit 1=Present 0=Not Present	TYPE	Type of Segment

Figure 4.9. 80286 Code and Data Segment Descriptors

By supporting 80286 system segments the Intel486 DX microprocessor is able to execute 80286 application programs on an Intel486 DX microprocessor operating system. This is possible because the processor automatically understands which descriptors are 80286-style descriptors and which descriptors are Intel486 DX Microprocessor-style descriptors. In particular, if the upper word of a descriptor is zero, then that descriptor is a 80286-style descriptor.

The only other differences between 80286-style descriptors and Intel486 DX microprocessor-style descriptors is the interpretation of the word count field of call gates and the B bit. The word count field specifies the number of 16-bit quantities to copy for 80286 call gates and 32-bit quantities for Intel486 DX microprocessor call gates. The B bit controls the size of PUSHes when using a call gate; if B=0 PUSHes are 16 bits, if B=1 PUSHes are 32 bits.

4.3.4.8 Selector Fields

A selector in Protected Mode has three fields: Local or Global Descriptor Table Indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 4.10. The TI bits select one of two memory-based tables of descriptors (the Global Descriptor Table or the Local Descriptor Table). The Index selects one of 8K descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

4.3.4.9 Segment Descriptor Cache

In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with that selector is automatically loaded (cached) on the chip. Once loaded, all references to that segment use the cached descriptor information instead of reaccessing the descriptor. The contents of the descriptor cache are not visible to the programmer. Since descriptor caches only change when a segment register is changed, programs which modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.

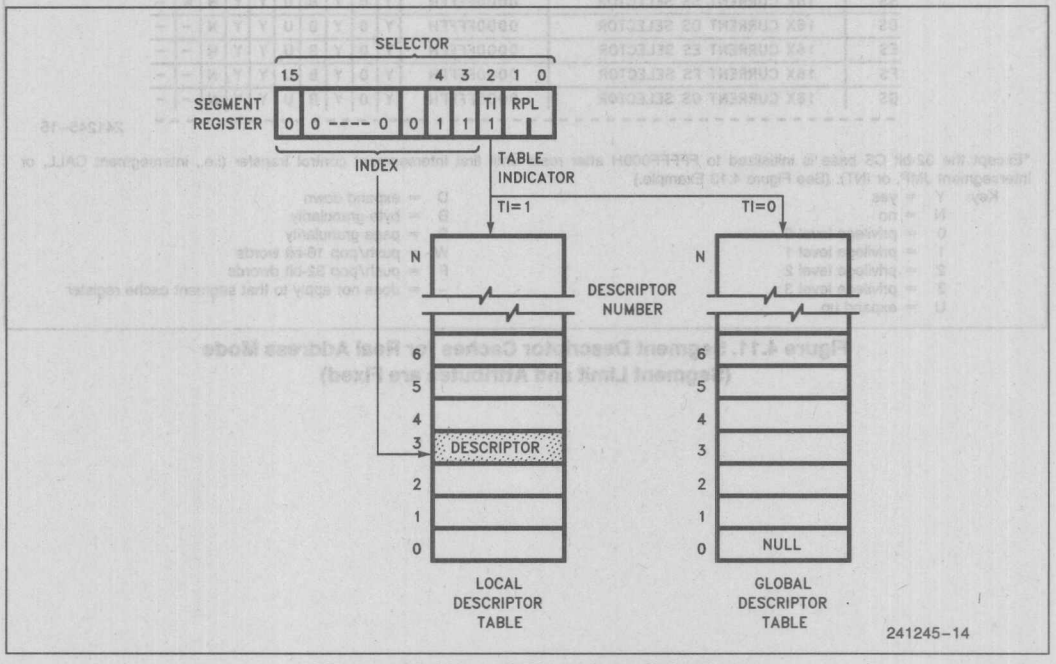
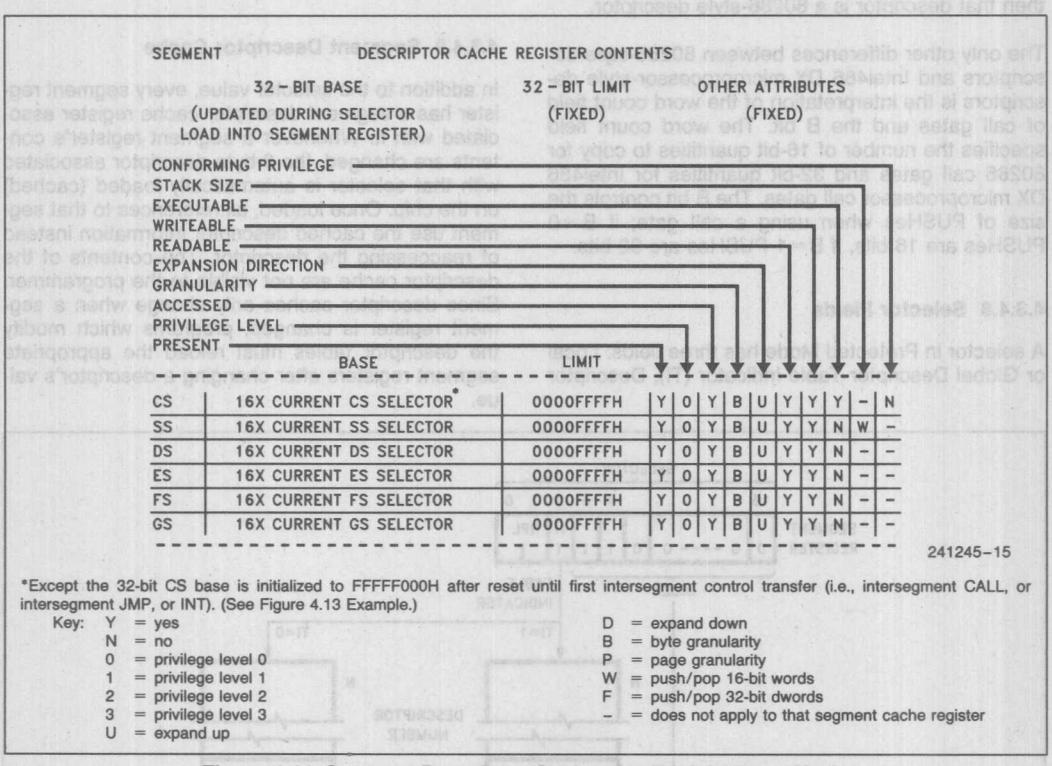


Figure 4.10. Example Descriptor Selection

4.3.4.10 Segment Descriptor Register Settings

The contents of the segment descriptor cache vary depending on the mode the Intel486 DX microprocessor is operating in. When operating in Real Address Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.11. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. In Real Address Mode, the internal "privilege level" is always fixed to the highest level, level 0, so I/O and other privileged opcodes may be executed.

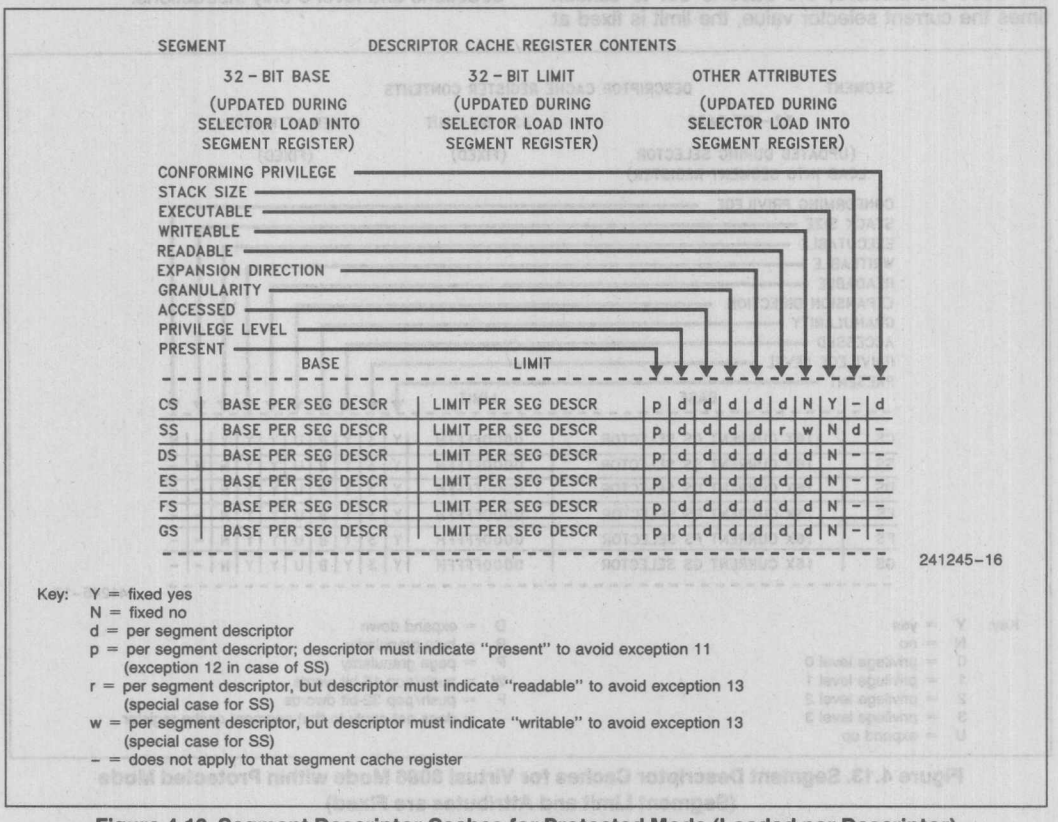


*Except the 32-bit CS base is initialized to FFFFF000H after reset until first intersegment control transfer (i.e., intersegment CALL, or intersegment JMP, or INT). (See Figure 4.13 Example.)

Key:
- Y = yes
- N = no
- 0 = privilege level 0
- 1 = privilege level 1
- 2 = privilege level 2
- 3 = privilege level 3
- U = expand up
- D = expand down
- B = byte granularity
- P = page granularity
- W = push/pop 16-bit words
- F = push/pop 32-bit dwords
- – = does not apply to that segment cache register

Figure 4.11. Segment Descriptor Caches for Real Address Mode (Segment Limit and Attributes are Fixed)

When operating in Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.12. In Protected Mode, each of these fields are defined according to the contents of the segment descriptor indexed by the selector value loaded into the segment register.



Key: Y = fixed yes
N = fixed no
d = per segment descriptor
p = per segment descriptor; descriptor must indicate "present" to avoid exception 11 (exception 12 in case of SS)
r = per segment descriptor, but descriptor must indicate "readable" to avoid exception 13 (special case for SS)
w = per segment descriptor, but descriptor must indicate "writable" to avoid exception 13 (special case for SS)
– = does not apply to that segment cache register

Figure 4.12. Segment Descriptor Caches for Protected Mode (Loaded per Descriptor)

When operating in a Virtual 8086 Mode within the Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.13. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. The virtual program executes at lowest privilege level, level 3, to allow trapping of all IOPL-sensitive instructions and level-0-only instructions.

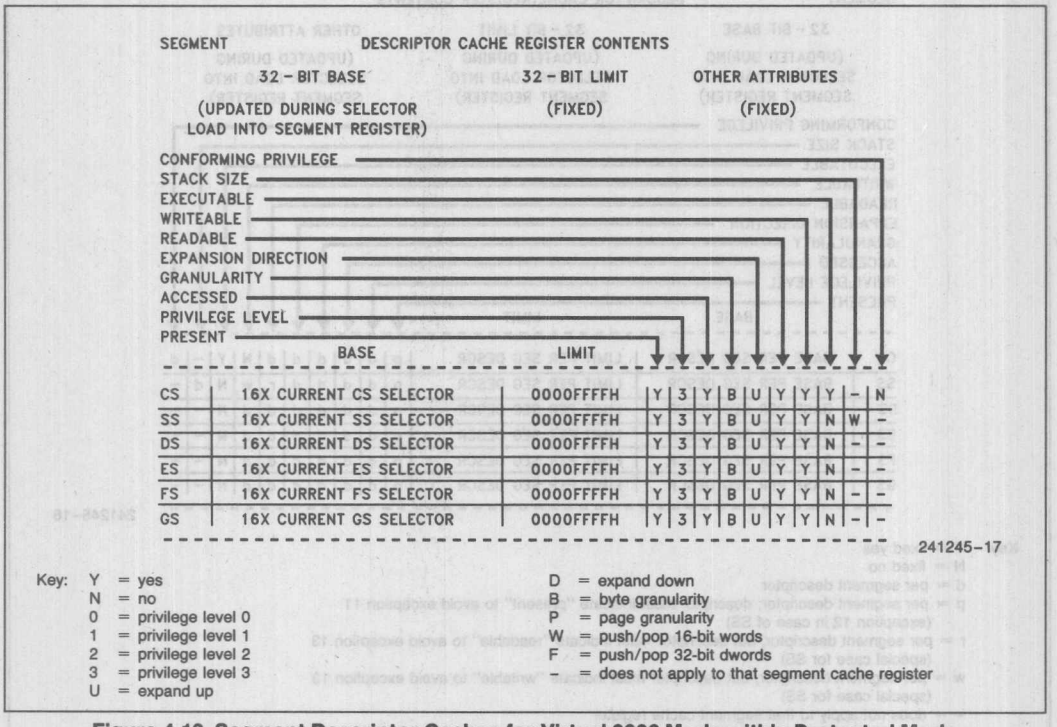


Figure 4.13. Segment Descriptor Caches for Virtual 8086 Mode within Protected Mode (Segment Limit and Attributes are Fixed)

4.4 Protection

4.4.1 PROTECTION CONCEPTS

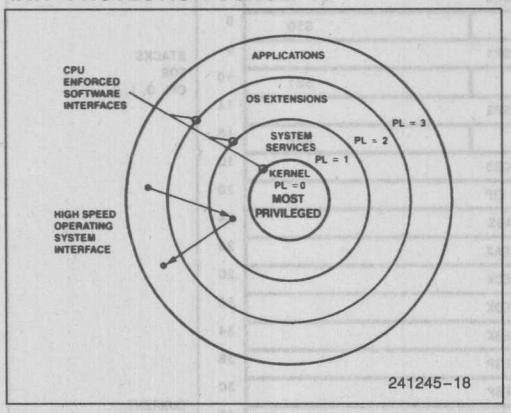


Figure 4.14. Four-Level Hierarchical Protection

The Intel486 DX Microprocessor has four levels of protection which are optimized to support the needs of a multi-tasking operating system to isolate and protect user programs from each other and the operating system. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. Unlike traditional microprocessor-based systems where this protection is achieved only through the use of complex external hardware and software the Intel486 DX Microprocessor provides the protection as part of its integrated Memory Management Unit. The Intel486 DX Microprocessor offers an additional type of protection on a page basis, when paging is enabled (See Section 4.5.3 **Page Level Protection**).

The four-level hierarchical privilege system is illustrated in Figure 4-14. It is an extension of the user/supervisor privilege mode commonly used by minicomputers and, in fact, the user/supervisor mode is fully supported by the Intel486 DX Microprocessor paging mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged or trusted level.

4.4.2 RULES OF PRIVILEGE

The Intel486 DX Microprocessor controls access to both data and procedures between levels of a task, according to the following rules.

- Data stored in a segment with privilege level **p** can be accessed only by code executing at a privilege level at least as privileged as **p**.
- A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p**.

4.4.3 PRIVILEGE LEVELS

4.4.3.1 Task Privilege

At any point in time, a task on the Intel486 DX Microprocessor always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies the task's privilege level. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level. (See Section 4.4.4 **Privilege Level Transfers**) Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

4.4.3.2 Selector Privilege (RPL)

The privilege level of a selector is specified by the RPL field. The RPL is the two least significant bits of the selector. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level for the use of a segment. This level is called the task's effective privilege level (EPL). The EPL is defined as being the least privileged (i.e. numerically larger) level of a task's CPL and a selector's RPL. Thus, if selector's RPL = 0 then the CPL always specifies the privilege level for making an access using the selector. On the other hand if RPL = 3 then a selector can only access segments at level 3 regardless of the task's CPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Since the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

4.4.3.3 I/O Privilege and I/O Permission Bitmap

The I/O privilege level (IOPL, a 2-bit field in the EFLAG register) defines the least privileged level at which I/O instructions can be unconditionally performed. I/O instructions can be unconditionally performed when CPL ≤ IOPL. (The I/O instructions are IN, OUT, INS, OUTS, REP INS, and REP OUTS.) When CPL > IOPL, and the current task is associated with a 286 TSS, attempted I/O instructions cause an exception 13 fault. When CPL > IOPL, and the current task is associated with an Intel486 DX Microprocessor TSS, the I/O Permission Bitmap (part of an Intel486 DX Microprocessor TSS) is consulted on whether I/O to the port is allowed, or an exception 13 fault is to be generated instead. For diagrams of the I/O Permission Bitmap, refer to Figures 4.15a and 4.15b. For further information on how the I/O Permission Bitmap is used in Protected Mode or in Virtual 8086 Mode, refer to Section 4.6.4 Protection and I/O Permission Bitmap.

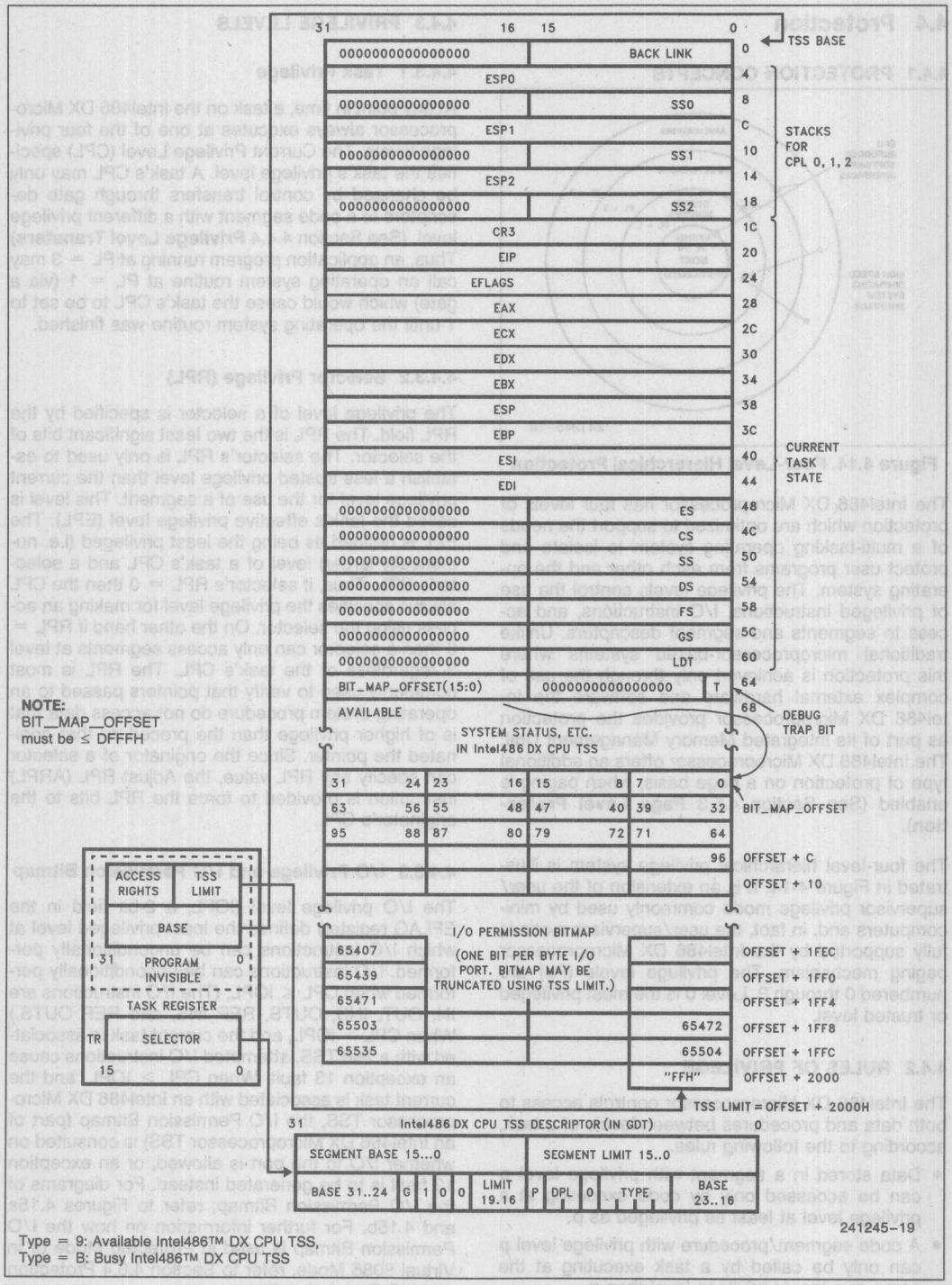


Figure 4.15a. Intel486™ DX Microprocessor TSS and TSS Registers

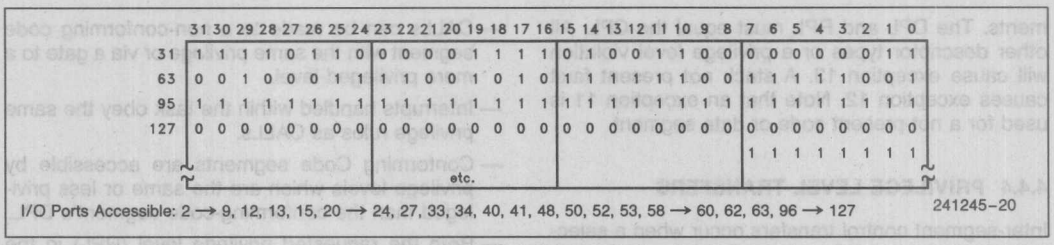


Figure 4.15b. Sample I/O Permission Bit Map

Table 4.2. Pointer Test Instructions

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

The I/O privilege level (IOPL) also affects whether several other instructions can be executed or cause an exception 13 fault instead. These instructions are called "IOPL-sensitive" instructions and they are CLI and STI. (Note that the LOCK prefix is *not* IOPL-sensitive on the Intel486 microprocessor.)

The IOPL also affects whether the IF (interrupts enable flag) bit can be changed by loading a value into the EFLAGS register. When CPL ≤ IOPL, then the IF bit can be changed by loading a new value into the EFLAGS register. When CPL > IOPL, the IF bit cannot be changed by a new value POP'ed into (or otherwise loaded into) the EFLAGS register; the IF bit merely remains unchanged and no exception is generated.

4.4.3.4 Privilege Validation

The Intel486 DX microprocessor provides several instructions to speed pointer testing and help maintain system integrity by verifying that the selector value refers to an appropriate segment. Table 4.2 summarizes the selector validation procedures available for the Intel486 DX microprocessor.

This pointer verification prevents the common problem of an application at PL = 3 calling a operating systems routine at PL = 0 and passing the operating system routine a "bad" pointer which corrupts a data structure belonging to the operating system. If the operating system routine uses the ARPL instruction to ensure that the RPL of the selector has no greater privilege than that of the caller, then this problem can be avoided.

4.4.3.5 Descriptor Access

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL as described above.

Any time an instruction loads data segment registers (DS, ES, FS, GS) the Intel486 DX microprocessor makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segments or readable code segments. The data access rules are specified in Section 4.4.2 **Rules of Privilege**. The only exception to those rules is readable conforming code segments which can be accessed at any privilege level.

Finally the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL an exception 13 (general protection fault) is generated.

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load selectors into SS must refer to data segment descriptors for writeable data seg-

ments. The DPL and RPL must equal the CPL. All other descriptor types or a privilege level violation will cause exception 13. A stack not present fault causes exception 12. Note that an exception 11 is used for a not-present code or data segment.

4.4.4 PRIVILEGE LEVEL TRANSFERS

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 4.3. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only via control transfers, by using gates, task switches, and interrupt or trap gates.

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13 (e.g. JMP through a call gate, or IRET from a normal subroutine call).

In order to provide further system security, all control transfers are also subject to the privilege rules.

The privilege rules require that:

— Privilege level transitions can only occur via gates.

— JMPs can be made to a non-conforming code segment with the same privilege or to a conforming code segment with greater or equal privilege.

— CALLs can be made to a non-conforming code segment with the same privilege or via a gate to a more privileged level.

— Interrupts handled within the task obey the same privilege rules as CALLs.

— Conforming Code segments are accessible by privilege levels which are the same or less privileged than the conforming-code segment's DPL.

— Both the requested privilege level (RPL) in the selector pointing to the gate and the task's CPL must be of equal or greater privilege than the gate's DPL.

— The code segment selected in the gate must be the same or more privileged than the task's CPL.

— Return instructions that do not switch tasks can only return control to a code segment with same or less privilege.

— Task switches can be performed by a CALL, JMP, or INT which references either a task gate or task state segment who's DPL is less privileged or the same privilege as the old task's CPL.

Any control transfer that changes CPL within a task causes a change of stacks as a result of the privilege level change. The initial values of SS:ESP for privilege levels 0, 1, and 2 are retained in the task state segment (see Section 4.4.6 **Task Switching**). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and ESP registers and the previous stack pointer is pushed onto the new stack.

Table 4.3. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level	CALL	Call Gate	GDT/LDT
Interrupt within task may change CPL	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
Task Switch	CALL, JMP	Task State Segment	GDT
	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag register) = 0
**NT (Nested Task bit of flag register) = 1

When RETurning to the original privilege level, use of the lower-privileged stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words (as specified in the gate's word count field) are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

4.4.5 CALL GATES

Gates provide protected, indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures (such as those which allocate memory, or perform I/O).

Gate descriptors follow the data access rules of privilege; that is, gates can be accessed by a task if the EPL, is equal to or more privileged than the gate descriptor's DPL. Gates follow the control transfer rules of privilege and therefore may only transfer control to a more privileged level.

Call Gates are accessed via a CALL instruction and are syntactically identical to calling a normal subroutine. When an inter-level Intel486 DX call gate is activated, the following actions occur.

1. Load CS:EIP from gate check for validity
2. SS is pushed zero-extended to 32 bits
3. ESP is pushed
4. Copy Word Count 32-bit parameters from the old stack to the new stack
5. Push Return address on stack

The procedure is identical for 80286 Call gates, except that 16-bit parameters are copied and 16-bit registers are pushed.

Interrupt Gates and Trap gates work in a similar fashion as the call gates, except there is no copying of parameters. The only difference between Trap and Interrupt gates is that control transfers through an Interrupt gate disable further interrupts (i.e. the IF bit is set to 0), and Trap gates leave the interrupt status unchanged.

4.4.6 TASK SWITCHING

A very important attribute of any multi-tasking/multi-user operating systems is its ability to rapidly switch between tasks or processes. The Intel486 DX microprocessor directly supports this operation by providing a task switch instruction in hardware. The Intel486 DX microprocessor task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task, in about 10 microseconds. Like transfer of control via gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 4.15) containing the entire Intel486 DX microprocessor execution state while a task gate descriptor contains a TSS selector. The Intel486 DX microprocessor supports both 80286 and Intel486 DX microprocessor style TSSs. Figure 4.16 shows a 80286 TSS. The limit of an Intel486 DX microprocessor TSS must be greater than 0064H (002BH for a 80286 TSS), and can be as large as 4 Gigabytes. In the additional TSS space, the operating system is free to store additional information such as the reason the task is inactive, time the task has spent running, and open files belong to the task.

Each task must have a TSS associated with it. The current TSS is identified by a special register in the Intel486 DX microprocessor called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The current executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word (CR0) give information about the state of a task which are useful to the operating system. The Nested Task (NT) (bit 14 in EFLAGS) controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular return; when NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

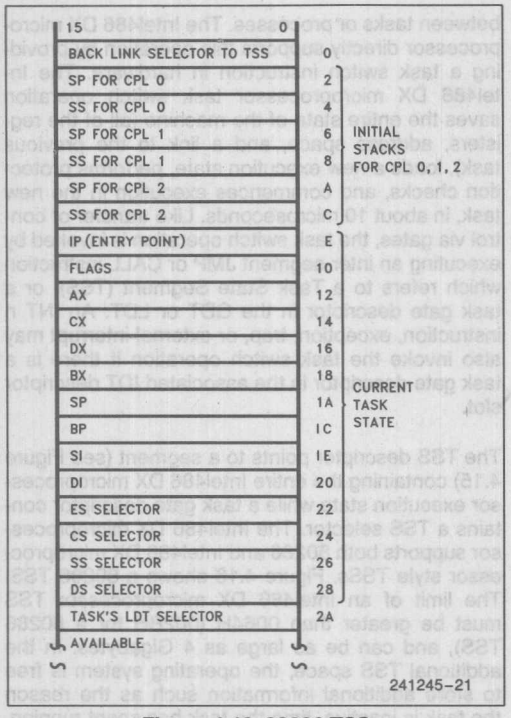


Figure 4.16. 80286 TSS

When a CALL or INT instruction initiates a task switch, the new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. (The NT bit will be restored after execution of the interrupt handler) NT may also be set or cleared by POPF or IRET instructions.

The Intel486 DX Microprocessor Task State Segment is marked busy by changing the descriptor type field from TYPE 9H to TYPE BH. An 80286 TSS is marked busy by changing the descriptor type field from TYPE 1 to TYPE 3. Use of a selector that references a busy task state segment causes an exception 13.

The Virtual Mode (VM) bit 17 is used to indicate if a task, is a virtual 8086 task. If VM = 1, then the tasks will use the Real Mode addressing mechanism. The virtual 8086 environment is only entered and exited via a task switch (see Section 4.6 **Virtual Mode**).

The FPU's state is not automatically saved when a task switch occurs, because the incoming task may not use the FPU. The Task Switched (TS) Bit (bit 3 in the CR0) helps deal with the FPU's state in a multitasking environment. Whenever the Intel486 DX Microprocessor switches tasks, it sets the TS bit. The Intel486 DX Microprocessor detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the FPU. A processor extension not present exception (7) will occur when attempting to execute a Floating Point or WAIT instruction if the Task Switched and Monitor coprocessor extension bits are both set (i.e. TS = 1 and MP = 1).

The **T** bit in the Intel486 DX Microprocessor TSS indicates that the processor should generate a debug exception when switching to a task. If T = 1 then upon entry to a new task a debug exception 1 will be generated.

4.4.7 INITIALIZATION AND TRANSITION TO PROTECTED MODE

Since the Intel486 DX Microprocessor begins executing in Real Mode immediately after RESET it is necessary to initialize the system tables and registers with the appropriate values.

The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256 bytes long, and GDT must contain descriptors for the initial code, and data segments. Figure 4.17 shows the tables and Figure 4.18 the descriptors needed for a simple Protected Mode Intel486 DX Microprocessor system. It has a single code and single data/stack segment each four gigabytes long and a single privilege level PL = 0.

The actual method of enabling Protected Mode is to load CR0 with the PE bit set, via the MOV CR0, R/M instruction. This puts the Intel486 DX Microprocessor in Protected Mode.

After enabling Protected Mode, the next instruction should execute an intersegment JMP to load the CS register and flush the instruction decode queue. The final step is to load all of the data segment registers with the initial selector values.

An alternate approach to entering Protected Mode which is especially appropriate for multi-tasking operating systems, is to use the built in task-switch to load all of the registers. In this case the GDT would contain two TSS descriptors in addition to the code and data descriptors needed for the first task. The first JMP instruction in Protected Mode would jump to the TSS causing a task switch and loading all of the registers with the values stored in the TSS. The Task State Segment Register should be initialized to point to a valid TSS descriptor since a task switch saves the state of the current task in a task state segment.

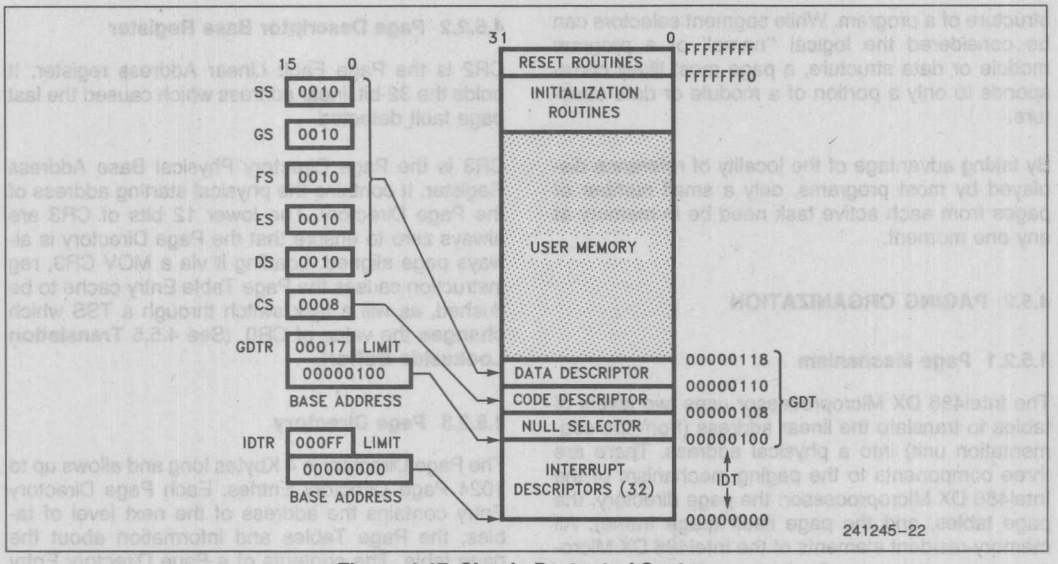


Figure 4.17. Simple Protected System

	2	BASE 31 . . . 24 00 (H)	G 1	D 1	0	0	LIMIT 19.16 F (H)	1	0	0	1	0	0	1	0	BASE 23 . . . 16 00 (H)
DATA DESCRIPTOR		SEGMENT BASE 15 . . . 0 0118 (H)						SEGMENT LIMIT 15 . . . 0 FFFF (H)									
	1	BASE 31 . . . 24 00 (H)	G 1	D 1	0	0	LIMIT 19.16 F (H)	1	0	0	1	1	0	1	0	BASE 23 . . . 16 00 (H)
CODE DESCRIPTOR		SEGMENT BASE 15 . . . 0 0118 (H)						SEGMENT LIMIT 15 . . . 0 FFFF (H)									
		NULL DESCRIPTOR															
	0																
		31					24						16	15		8	0

Figure 4.18. GDT Descriptors for Simple System

4.4.8 TOOLS FOR BUILDING PROTECTED SYSTEMS

In order to simplify the design of a protected multitasking system, Intel provides a tool which allows the system designer an easy method of constructing the data structures needed for a Protected Mode Intel486 DX microprocessor system. This tool is the builder BLD-386. BLD-386 lets the operating system writer specify all of the segment descriptors discussed in the previous sections (LDTs, IDTs, GDTs, Gates, and TSSs) in a high-level language.

4.5 Paging

4.5.1 PAGING CONCEPTS

Paging is another type of memory management useful for virtual memory multitasking operating systems. Unlike segmentation which modularizes programs and data into variable length segments, paging divides programs into multiple uniform size pages. Pages bear no direct relation to the logical

structure of a program. While segment selectors can be considered the logical "name" of a program module or data structure, a page most likely corresponds to only a portion of a module or data structure.

By taking advantage of the locality of reference displayed by most programs, only a small number of pages from each active task need be in memory at any one moment.

4.5.2 PAGING ORGANIZATION

4.5.2.1 Page Mechanism

The Intel486 DX Microprocessor uses two levels of tables to translate the linear address (from the segmentation unit) into a physical address. There are three components to the paging mechanism of the Intel486 DX Microprocessor: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the Intel486 DX Microprocessor paging mechanism are the same size, namely, 4 Kbytes. A uniform size for all of the elements simplifies memory allocation and reallocation schemes, since there is no problem with memory fragmentation. Figure 4.19 shows how the paging mechanism works.

4.5.2.2 Page Descriptor Base Register

CR2 is the Page Fault Linear Address register. It holds the 32-bit linear address which caused the last page fault detected.

CR3 is the Page Directory Physical Base Address Register. It contains the physical starting address of the Page Directory. The lower 12 bits of CR3 are always zero to ensure that the Page Directory is always page aligned. Loading it via a MOV CR3, reg instruction causes the Page Table Entry cache to be flushed, as will a task switch through a TSS which **changes** the value of CR0. (See 4.5.5 **Translation Lookaside Buffer**).

4.5.2.3 Page Directory

The Page Directory is 4 Kbytes long and allows up to 1024 Page Directory Entries. Each Page Directory Entry contains the address of the next level of tables, the Page Tables and information about the page table. The contents of a Page Directory Entry are shown in Figure 4.20. The upper 10 bits of the linear address (A22–A31) are used as an index to select the correct Page Directory Entry.

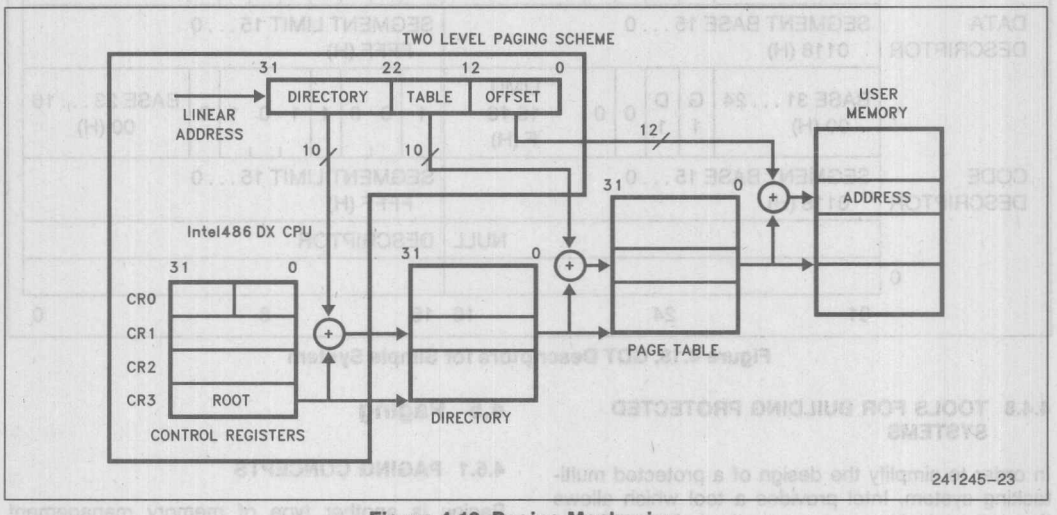


Figure 4.19. Paging Mechanism

31 12	11 10	9	8	7	6	5	4	3	2	1	0
PAGE TABLE ADDRESS 31..12	OS RESERVED		0	0	D	A	PCD	PWT	U/S	R/W	P

Figure 4.20. Page Directory Entry (Points to Page Table)

31 12	11 10	9	8	7	6	5	4	3	2	1	0
PAGE FRAME ADDRESS 31..12	OS RESERVED		0	0	D	A	PCD	PWT	U/S	R/W	P

Figure 4.21. Page Table Entry (Points to Page)

4.5.2.4 Page Tables

Each Page Table is 4 Kbytes and holds up to 1024 Page Table Entries. Page Table Entries contain the starting address of the page frame and statistical information about the page (see Figure 4.21). Address bits A12–A21 are used as an index to select one of the 1024 Page Table Entries. The 20 upper-bit page frame address is concatenated with the lower 12 bits of the linear address to form the physical address. Page tables can be shared between tasks and swapped to disks.

4.5.2.5 Page Directory/Table Entries

The lower 12 bits of the Page Table Entries and Page Directory Entries contain statistical information about pages and page tables respectively. The **P** (Present) bit 0 indicates if a Page Directory or Page Table entry can be used in address translation. If P = 1 the entry can be used for address translation if P = 0 the entry can not be used for translation, and all of the other bits are available for use by the software. For example the remaining 31 bits could be used to indicate where on the disk the page is stored.

The **A** (Accessed) bit 5, is set by the Intel486 DX microprocessor for both types of entries before a read or write access occurs to an address covered by the entry. The **D** (Dirty) bit 6 is set to 1 before a write to an address covered by that page table entry occurs. The D bit is undefined for Page Directory Entries. When the P, A and D bits are updated by the Intel486 DX microprocessor, the processor generates a Read-Modify-Write cycle which locks the bus and prevents conflicts with other processors or perpherials. Software which modifies these bits should use the LOCK prefix to ensure the integrity of the page tables in multi-master systems.

The 3 bits marked **OS Reserved** in Figure 4.20 and Figure 4.21 (bits 9–11) are software definable. OSs are free to use these bits for whatever purpose they wish. An example use of the **OS Reserved** bits would be to store information about page aging. By keeping track of how long a page has been in memory since being accessed, an operating system can implement a page replacement algorithm like Least Recently Used.

The (User/Supervisor) U/S bit 2 and the (Read/Write) R/W bit 1 are used to provide protection attributes for individual pages.

4.5.3 PAGE LEVEL PROTECTION (R/W, U/S BITS)

The Intel486 DX microprocessor provides a set of protection attributes for paging systems. The paging mechanism distinguishes between two levels of protection: User which corresponds to level 3 of the segmentation based protection, and supervisor which encompasses all of the other protection levels (0, 1, 2).

The R/W and U/S bits are used in conjunction with the WP bit in the flags register (EFLAGS). The Intel386 microprocessor does not contain the WP bit. The WP bit has been added to the Intel486 DX microprocessor to protect read-only pages from supervisor write accesses. The Intel386 microprocessor allows a read-only page to be written from protection levels 0, 1 or 2. WP=0 is the Intel386 microprocessor compatible mode. When WP=0 the supervisor can write to a read-only page as defined by the U/S and R/W bits. When WP=1 supervisor access to a read-only page (R/W=0) will cause a page fault (exception 14).

Table 4.4 shows the affect of the WP, U/S and R/W bits on accessing memory. When WP=0, the supervisor can write to pages regardless of the state of the R/W bit. When WP=1 and R/W=0 the supervisor cannot write to a read-only page. A user attempt to access a supervisor only page (U/S=0), or write to a read only page will cause a page fault (exception 14).

The R/W and U/S bits provide protection from user access on a page by page basis since the bits are contained in the Page Table Entry and the Page Directory Table. The U/S and R/W bits in the first level Page Directory Table apply to all entries in the page table pointed to by that directory entry. The U/S and R/W bits in the second level Page Table Entry apply only to the page described by that entry. The most restrictive of the U/S and R/W bits from the Page Directory Table and the Page Table Entry are used to address a page.

Example: If the U/S and R/W bits for the Page Directory entry were 10 (user read/execute) and the

U/S and R/W bits for the Page Table Entry were 01 (no user access at all), the access rights for the page would be 01, the numerically smaller of the two.

Note that a given segment can be easily made read-only for level 0, 1 or 2 via use of segmented protection mechanisms. (Section 4.4 **Protection**).

4.5.4 PAGE CACHEABILITY (PWT AND PCD BITS)

PWT (page write through) and PCD (page cache disable) are two new bits defined in entries in both levels of the page table structure, the Page Directory Table and the Page Table Entry. PCD and PWT control page cacheability and write policy.

PWT controls write policy. PWT = 1 defines a write-through policy for the current page. PWT = 0 allows the possibility of write-back. PWT is ignored internally because the Intel486 DX microprocessor has a write-through cache. PWT can be used to control the write policy of a second level cache.

PCD controls cacheability. PCD = 0 enables caching in the on-chip cache. PCD alone does not enable caching, it must be conditioned by the KEN# (cache enable) input signal and the state of the CD (cache disable bit) and NW (no write-through) bits in control register 0 (CR0). When PCD = 1, caching is disabled regardless of the state of KEN#, CD and NW. (See Section 5.0, **On-Chip Cache**).

The state of the PCD and PWT bits are driven out on the PCD and PWT pins during a memory access.

The PWT and PCD bits for a bus cycle are obtained either from control register 3 (CR3), the Page Directory Entry or the Page Table Entry, depending on the type of cycle run. However, when paging is disabled (PG = 0 in CR0) or for cycles which bypass paging (i.e., I/O (input/output) references, INTR (interrupt request) and HALT cycles), the PCD and PWT bits of CR3 are ignored. The Intel486 DX CPU assumes PCD = 0 and PWT = 0 and drives these values on the PCD and PWT pins.

When paging is enabled (PG = 1 in CR0), the bits from the page table entry are cached in the translation lookaside buffer (TLB), and are driven any time the page mapped by the TLB entry is referenced. For normal memory cycles run with paging enabled, the PWT and PCD bits are taken from the Page Table Entry. During TLB refresh cycles when the Page Directory and Page Table entries are read, the PWT and PCD bits must be obtained elsewhere. The bits are taken from CR3 when a Page Directory Entry is being read. The bits are taken from the Page Directory Entry when the Page Table Entry is being updated.

The PCD or PWT bits in CR3 are initialized to zero at reset, but can be set to any value by level 0 software.

4.5.5 TRANSLATION LOOKASIDE BUFFER

The Intel486 DX Microprocessor paging hardware is designed to support demand paged virtual memory systems. However, performance would degrade substantially if the processor was required to access two levels of tables for every memory reference. To solve this problem, the Intel486 DX Microprocessor keeps a cache of the most recently accessed pages, this cache is called the Translation Lookaside Buffer (TLB). The TLB is a four-way set associative 32-entry page table cache. It automatically keeps the most commonly used Page Table Entries in the processor. The 32-entry TLB coupled with a 4K page size, results in coverage of 128 Kbytes of memory addresses. For many common multi-tasking systems, the TLB will have a hit rate of about 98%. This means that the processor will only have to access the two-level page structure on 2% of all memory references. Figure 4.22 illustrates how the TLB complements the Intel486 DX Microprocessor's paging mechanism.

Reading a new entry into the TLB (TLB refresh) is a two step process handled by the Intel486 DX microprocessor hardware. The sequence of data cycles to perform a TLB refresh are:

Table 4.4. Page Level Protection Attributes

U/S	R/W	WP	User Access	Supervisor Access
0	0	0	None	Read/Write/Execute
0	1	0	None	Read/Write/Execute
1	0	0	Read/Execute	Read/Write/Execute
1	1	0	Read/Write/Execute	Read/Write/Execute
0	0	1	None	Read/Execute
0	1	1	None	Read/Write/Execute
1	0	1	Read/Execute	Read/Execute
1	1	1	Read/Write/Execute	Read/Write/Execute

1. Read the correct Page Directory Entry, as pointed to by the page base register and the upper 10 bits of the linear address. The page base register is in control register 3.

1a. Optionally perform a locked read/write to set the accessed bit in the directory entry. The directory entry will actually get read twice if the Intel486 DX microprocessor needs to set any of the bits in the entry. If the page directory entry changes between the first and second reads, the data returned for the second read will be used.

2. Read the correct entry in the Page Table and place the entry in the TLB.

2a. Optionally perform a locked read/write to set the accessed and/or dirty bit in the page table entry. Again, note that the page table entry will actually get read twice if the Intel486 DX microprocessor needs to set any of the bits in the entry. Like the directory entry, if the data changes between the first and second read the data returned for the second read will be used.

Note that the directory entry must always be read into the processor, since directory entries are never placed in the paging TLB. Page faults can be signaled from either the page directory read or the page table read. Page directory and page table entries may be placed in the Intel486 DX on-chip cache just like normal data.

4.5.6 PAGING OPERATION

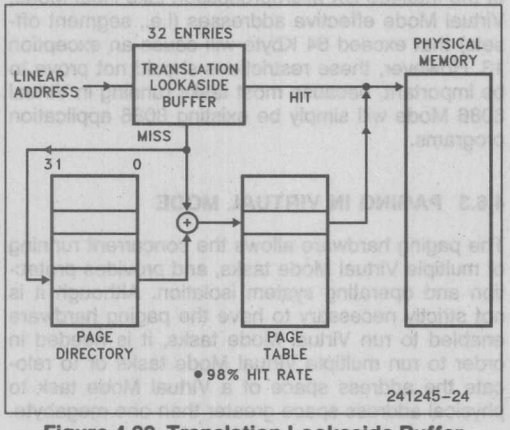


Figure 4.22. Translation Lookaside Buffer

The paging hardware operates in the following fashion. The paging unit hardware receives a 32-bit linear address from the segmentation unit. The upper 20 linear address bits are compared with all 32 entries in the TLB to determine if there is a match. If there is a match (i.e., a TLB hit), then the 32-bit physical address is calculated and will be placed on the address bus.

However, if the page table entry is not in the TLB, the Intel486 DX Microprocessor will read the appropriate Page Directory Entry. If P = 1 on the Page Directory Entry indicating that the page table is in memory, then the Intel486 DX Microprocessor will read the appropriate Page Table Entry and set the Access bit. If P = 1 on the Page Table Entry indicating that the page is in memory, the Intel486 DX Microprocessor will update the Access and Dirty bits as needed and fetch the operand. The upper 20 bits of the linear address, read from the page table, will be stored in the TLB for future accesses. However, if P = 0 for either the Page Directory Entry or the Page Table Entry, then the processor will generate a page fault, an Exception 14.

The processor will also generate an exception 14 page fault, if the memory reference violated the page protection attributes (i.e., U/S or R/W) (e.g., trying to write to a read-only page). CR2 will hold the linear address which caused the page fault. If a second page fault occurs, while the processor is attempting to enter the service routine for the first, then the processor will invoke the page fault (exception 14) handler a second time, rather than the double fault (exception 8) handler. Since Exception 14 is classified as a fault, CS: EIP will point to the instruction causing the page fault. The 16-bit error code pushed as part of the page fault handler will contain status bits which indicate the cause of the page fault.

The 16-bit error code is used by the operating system to determine how to handle the page fault. Figure 4.23a shows the format of the page-fault error code and the interpretation of the bits.

NOTE:
Even though the bits in the error code (U/S, W/R, and P) have similar names as the bits in the Page Directory/Table Entries, the interpretation of the error code bits is different. Figure 4.23b indicates what type of access caused the page fault.

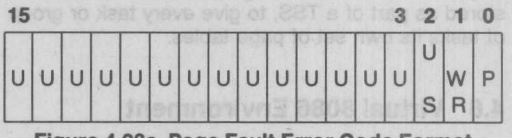


Figure 4.23a. Page Fault Error Code Format

U/S: The U/S bit indicates whether the access causing the fault occurred when the processor was executing in User Mode (U/S = 1) or in Supervisor mode (U/S = 0).

W/R: The W/R bit indicates whether the access causing the fault was a Read (W/R = 0) or a Write (W/R = 1).

P: The P bit indicates whether a page fault was caused by a not-present page (P = 0), or by a page level protection violation (P = 1).

U: UNDEFINED

U/S	W/R	Access Type
0	0	Supervisor* Read
0	1	Supervisor Write
1	0	User Read
1	1	User Write

*Descriptor table access will fault with U/S = 0, even if the program is executing at level 3.

Figure 4.23b. Type of Access Causing Page Fault

4.5.7 OPERATING SYSTEM RESPONSIBILITIES

The Intel486 DX Microprocessor takes care of the page address translation process, relieving the burden from an operating system in a demand-paged system. The operating system is responsible for setting up the initial page tables, and handling any page faults. The operating system also is required to invalidate (i.e., flush) the TLB when any changes are made to any of the page table entries. The operating system must reload CR3 to cause the TLB to be flushed.

Setting up the tables is simply a matter of loading CR3 with the address of the Page Directory, and allocating space for the Page Directory and the Page Tables. The primary responsibility of the operating system is to implement a swapping policy and handle all of the page faults.

A final concern of the operating system is to ensure that the TLB cache matches the information in the paging tables. In particular, any time the operating system sets the P present bit of page table entry to zero, the TLB must be flushed. Operating systems may want to take advantage of the fact that CR3 is stored as part of a TSS, to give every task or group of tasks its own set of page tables.

4.6 Virtual 8086 Environment

4.6.1 EXECUTING 8086 PROGRAMS

The Intel486 DX Microprocessor allows the execution of 8086 application programs in both Real Mode and in the Virtual 8086 Mode (Virtual Mode). Of the two methods, Virtual 8086 Mode offers the system designer the most flexibility. The Virtual 8086 Mode allows the execution of 8086 applications, while still allowing the system designer to take full advantage of the Intel486 DX Microprocessor protection mechanism. In particular, the Intel486 DX Microprocessor allows the simultaneous execution of 8086 operating systems and its applications, and an Intel486 DX Microprocessor operating system and both 80286 and Intel486 DX Microprocessor applications. Thus, in a multi-user Intel486 DX Microprocessor computer, one person could be running an MS-DOS spreadsheet, another person using MS-DOS, and a third person could be running multiple Unix utilities and applications. Each person in this scenario would believe that he had the computer completely to himself. Figure 4.24 illustrates this concept.

4.6.2 VIRTUAL 8086 MODE ADDRESSING MECHANISM

One of the major differences between Intel486 DX Microprocessor Real and Protected modes is how the segment selectors are interpreted. When the processor is executing in Virtual 8086 Mode the segment registers are used in an identical fashion to Real Mode. The contents of the segment register is shifted left 4 bits and added to the offset to form the segment base linear address.

The Intel486 DX Microprocessor allows the operating system to specify which programs use the 8086 style address mechanism, and which programs use Protected Mode addressing, on a per task basis. Through the use of paging, the one megabyte address space of the Virtual Mode task can be mapped to anywhere in the 4 gigabyte linear address space of the Intel486 DX Microprocessor. Like Real Mode, Virtual Mode effective addresses (i.e., segment offsets) that exceed 64 Kbyte will cause an exception 13. However, these restrictions should not prove to be important, because most tasks running in Virtual 8086 Mode will simply be existing 8086 application programs.

4.6.3 PAGING IN VIRTUAL MODE

The paging hardware allows the concurrent running of multiple Virtual Mode tasks, and provides protection and operating system isolation. Although it is not strictly necessary to have the paging hardware enabled to run Virtual Mode tasks, it is needed in order to run multiple Virtual Mode tasks or to relocate the address space of a Virtual Mode task to physical address space greater than one megabyte.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into up to 256 pages. Each one of the pages can be located anywhere within the maximum 4 gigabyte physical address space of the Intel486 DX Microprocessor. In addition, since CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations.

Finally, the paging hardware allows the sharing of the 8086 operating system code between multiple 8086 applications. Figure 4.24 shows how the Intel486 DX Microprocessor paging hardware enables multiple 8086 programs to run under a virtual memory demand paged system.

4.6.4 PROTECTION AND I/O PERMISSION BITMAP

All Virtual 8086 Mode programs execute at privilege level 3, the level of least privilege. As such, Virtual 8086 Mode programs are subject to all of the protection checks defined in Protected Mode. (This is different from Real Mode which implicitly is executing at privilege level 0, the level of greatest privilege.) Thus, an attempt to execute a privileged instruction when in Virtual 8086 Mode will cause an exception 13 fault.

The following are privileged instructions, which may be executed only at Privilege Level 0. Therefore, attempting to execute these instructions in Virtual 8086 Mode (or anytime CPL > 0) causes an exception 13 fault:

```
LIDT;  MOV DRn,reg;  MOV reg,DRn;
LGDT;  MOV TRn,reg;  MOV reg,TRn;
LMSW;  MOV CRn,reg;  MOV reg,CRn.
CLTS;
HLT;
```

Several instructions, particularly those applying to the multitasking model and protection model, are available only in Protected Mode. Therefore, attempting to execute the following instructions in Real Mode or in Virtual 8086 Mode generates an exception 6 fault:

```
LTR;   STR;
LLDT;  SLDT;
LAR;   VERR;
LSL;   VERW;
ARPL.
```

The instructions which are IOPL-sensitive in Protected Mode are:

```
IN;        STI;
OUT;       CLI
INS;
OUTS;
REP INS;
REP OUTS;
```

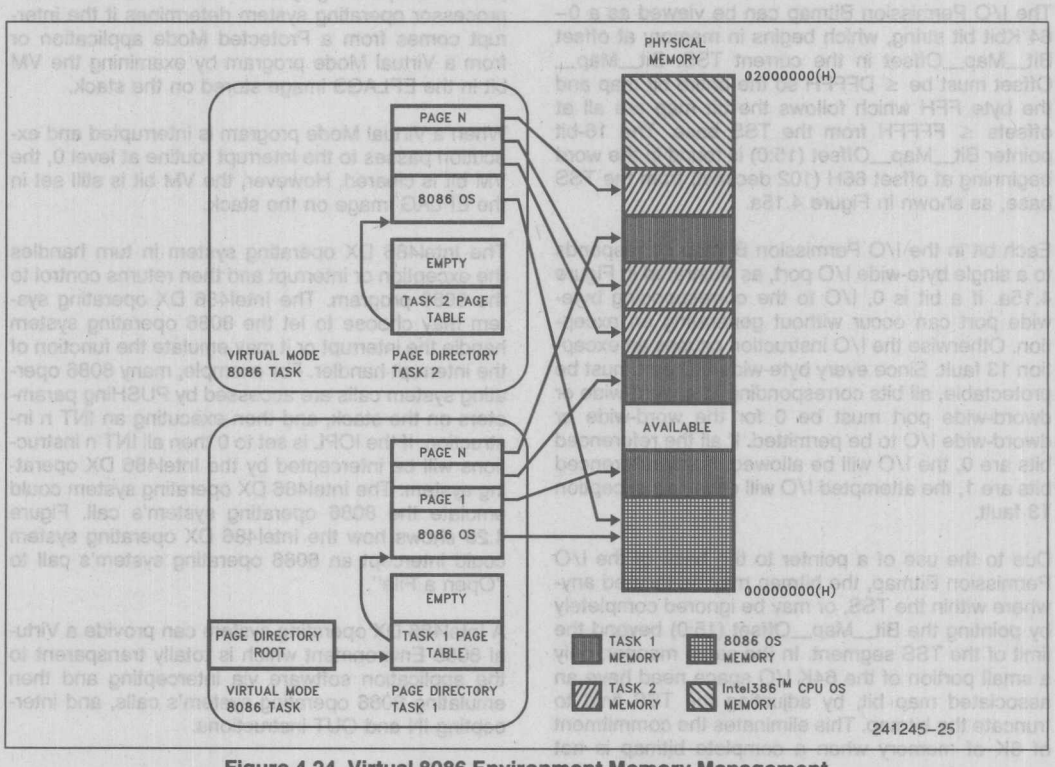


Figure 4.24. Virtual 8086 Environment Memory Management

In Virtual 8086 Mode, a slightly different set of instructions are made IOPL-sensitive. The following instructions are IOPL-sensitive in Virtual 8086 Mode:

```
INT n;    STI;
PUSHF;    CLI;
POPF;     IRET
```

The PUSHF, POPF, and IRET instructions are IOPL-sensitive in Virtual 8086 Mode only. This provision allows the IF flag (interrupt enable flag) to be virtualized to the Virtual 8086 Mode program. The INT n software interrupt instruction is also IOPL-sensitive in Virtual 8086 Mode. Note, however, that the INT 3 (opcode 0CCH), INTO, and BOUND instructions are not IOPL-sensitive in Virtual 8086 mode (they aren't IOPL sensitive in Protected Mode either).

Note that the I/O instructions (IN, OUT, INS, OUTS, REP INS, and REP OUTS) are **not** IOPL-sensitive in Virtual 8086 mode. Rather, the I/O instructions become automatically sensitive to the **I/O Permission Bitmap** contained in the **Intel486 DX Microprocessor Task State Segment**. The I/O Permission Bitmap, automatically used by the Intel486 DX microprocessor in Virtual 8086 Mode, is illustrated by Figures 4.15a and 4.15b.

The I/O Permission Bitmap can be viewed as a 0–64 Kbit bit string, which begins in memory at offset Bit_Map_Offset in the current TSS. Bit_Map_Offset must be ≤ DFFFH so the entire bit map and the byte FFH which follows the bit map are all at offsets ≤ FFFFH from the TSS base. The 16-bit pointer Bit_Map_Offset (15:0) is found in the word beginning at offset 66H (102 decimal) from the TSS base, as shown in Figure 4.15a.

Each bit in the I/O Permission Bitmap corresponds to a single byte-wide I/O port, as illustrated in Figure 4.15a. If a bit is 0, I/O to the corresponding byte-wide port can occur without generating an exception. Otherwise the I/O instruction causes an exception 13 fault. Since every byte-wide I/O port must be protectable, all bits corresponding to a word-wide or dword-wide port must be 0 for the word-wide or dword-wide I/O to be permitted. If all the referenced bits are 0, the I/O will be allowed. If any referenced bits are 1, the attempted I/O will cause an exception 13 fault.

Due to the use of a pointer to the base of the I/O Permission Bitmap, the bitmap may be located anywhere within the TSS, or may be ignored completely by pointing the Bit_Map_Offset (15:0) beyond the limit of the TSS segment. In the same manner, only a small portion of the 64K I/O space need have an associated map bit, by adjusting the TSS limit to truncate the bitmap. This eliminates the commitment of 8K of memory when a complete bitmap is not required, while allowing the fully general case if desired.

EXAMPLE OF BITMAP FOR I/O PORTS 0–255: Setting the TSS limit to {bit_Map_Offset + 31 +1**} [** see note below] will allow a 32-byte bitmap for the I/O ports #0–255, plus a terminator byte of all 1's [** see note below]. This allows the I/O bitmap to control I/O Permission to I/O port 0–255 while causing an exception 13 fault on attempted I/O to any I/O port 80256 through 65,565.

****IMPORTANT IMPLEMENTATION NOTE:** Beyond the last byte of I/O mapping information in the I/O Permission Bitmap **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the Intel486 microprocessor TSS segment (see Figure 4.15a).

4.6.5 INTERRUPT HANDLING

In order to fully support the emulation of an 8086 machine, interrupts in Virtual 8086 Mode are handled in a unique fashion. When running in Virtual Mode all interrupts and exceptions involve a privilege change back to the host Intel486 DX microprocessor operating system. The Intel486 DX microprocessor operating system determines if the interrupt comes from a Protected Mode application or from a Virtual Mode program by examining the VM bit in the EFLAGS image stored on the stack.

When a Virtual Mode program is interrupted and execution passes to the interrupt routine at level 0, the VM bit is cleared. However, the VM bit is still set in the EFLAG image on the stack.

The Intel486 DX operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The Intel486 DX operating system may choose to let the 8086 operating system handle the interrupt or it may emulate the function of the interrupt handler. For example, many 8086 operating system calls are accessed by PUSHing parameters on the stack, and then executing an INT n instruction. If the IOPL is set to 0 then all INT n instructions will be intercepted by the Intel486 DX operating system. The Intel486 DX operating system could emulate the 8086 operating system's call. Figure 4.25 shows how the Intel486 DX operating system could intercept an 8086 operating system's call to "Open a File".

A Intel486 DX operating system can provide a Virtual 8086 Environment which is totally transparent to the application software via intercepting and then emulating 8086 operating system's calls, and intercepting IN and OUT instructions.

4.6.6 ENTERING AND LEAVING VIRTUAL 8086 MODE

Virtual 8086 mode is entered by executing an IRET instruction (at CPL = 0), or Task Switch (at any CPL) to a Intel486 DX task whose Intel486 DX TSS has a FLAGS image containing a 1 in the VM bit position while the processor is executing in Protected Mode. That is, one way to enter Virtual 8086 mode is to switch to a task with a Intel486 DX TSS that has a 1 in the VM bit in the EFLAGS image. The other way is to execute a 32-bit IRET instruction at privilege level 0, where the stack has a 1 in the VM bit in the EFLAGS image. POPF does not affect the VM bit, even if the processor is in Protected Mode or level 0, and so cannot be used to enter Virtual 8086 Mode. PUSHF always pushes a 0 in the VM bit, even if the processor is in Virtual 8086 Mode, so that a program cannot tell if it is executing in REAL mode, or in Virtual 8086 mode.

The VM bit can be set by executing an IRET instruction only at privilege level 0, or by any instruction or Interrupt which causes a task switch in Protected Mode (with VM = 1 in the new FLAGS image), and can be cleared only by an interrupt or exception in Virtual 8086 Mode. IRET and POPF instructions executed in REAL mode or Virtual 8086 mode will not change the value in the VM bit.

The transition out of virtual 8086 mode to Intel486 DX protected mode occurs only on receipt of an interrupt or exception (such as due to a sensitive instruction). In Virtual 8086 mode, all interrupts and exceptions vector through the protected mode IDT, and enter an interrupt handler in protected Intel486 DX mode. That is, as part of interrupt processing, the VM bit is cleared.

Because the matching IRET must occur from level 0, if an Interrupt or Trap Gate is used to field an interrupt or exception out of Virtual 8086 mode, the Gate must perform an inter-level interrupt only to level 0. Interrupt or Trap Gates through conforming segments, or through segments with DPL > 0, will raise a GP fault with the CS selector as the error code.

4.6.6.1 Task Switches To/From Virtual 8086 Mode

Tasks which can execute in virtual 8086 mode must be described by a TSS with the Intel486 DX Microprocessor format (TYPE 9 or 11 descriptor).

A task switch out of virtual 8086 mode will operate exactly the same as any other task switch out of a task with an Intel486 DX TSS. All of the programmer visible state, including the FLAGS register with the VM bit set to 1, is stored in the TSS.

The segment registers in the TSS will contain 8086 segment base values rather than selectors.

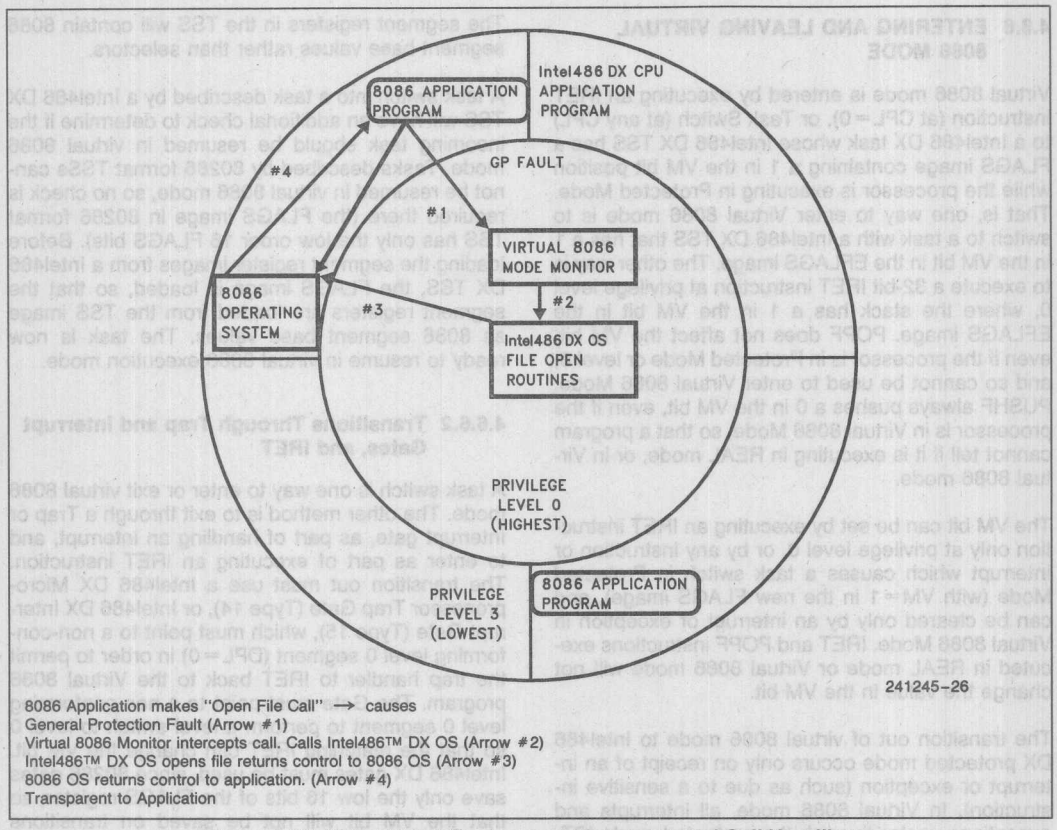
A task switch into a task described by a Intel486 DX TSS will have an additional check to determine if the incoming task should be resumed in virtual 8086 mode. Tasks described by 80286 format TSSs cannot be resumed in virtual 8086 mode, so no check is required there (the FLAGS image in 80286 format TSS has only the low order 16 FLAGS bits). Before loading the segment register images from a Intel486 DX TSS, the FLAGS image is loaded, so that the segment registers are loaded from the TSS image as 8086 segment base values. The task is now ready to resume in virtual 8086 execution mode.

4.6.6.2 Transitions Through Trap and Interrupt Gates, and IRET

A task switch is one way to enter or exit virtual 8086 mode. The other method is to exit through a Trap or Interrupt gate, as part of handling an interrupt, and to enter as part of executing an IRET instruction. The transition out must use a Intel486 DX Microprocessor Trap Gate (Type 14), or Intel486 DX Interrupt Gate (Type 15), which must point to a non-conforming level 0 segment (DPL = 0) in order to permit the trap handler to IRET back to the Virtual 8086 program. The Gate must point to a non-conforming level 0 segment to perform a level switch to level 0 so that the matching IRET can change the VM bit. Intel486 DX gates must be used, since 80286 gates save only the low 16 bits of the FLAGS register, so that the VM bit will not be saved on transitions through the 80286 gates. Also, the 16-bit IRET (presumably) used to terminate the 80286 interrupt handler will pop only the lower 16 bits from FLAGS, and will not affect the VM bit. The action taken for a Intel486 DX Trap or Interrupt gate if an interrupt occurs while the task is executing in virtual 8086 mode is given by the following sequence.

(1) Save the FLAGS register in a temp to push later. Turn off the VM and TF bits, and if the interrupt is serviced by an Interrupt Gate, turn off IF also.

(2) Interrupt and Trap gates must perform a level switch from 3 (where the VM86 program executes) to level 0 (so IRET can return). This process involves a stack switch to the stack given in the TSS for privilege level 0. Save the Virtual 8086 Mode SS and ESP registers to push in a later step. The segment register load of SS will be done as a Protected Mode segment load, since the VM bit was turned off above.



8086 Application makes "Open File Call" → causes
General Protection Fault (Arrow #1)
Virtual 8086 Monitor intercepts call. Calls Intel486™ DX OS (Arrow #2)
Intel486™ DX OS opens file returns control to 8086 OS (Arrow #3)
8086 OS returns control to application. (Arrow #4)
Transparent to Application

Figure 4.25. Virtual 8086 Environment Interrupt and Call Handling

(3) Push the 8086 segment register values onto the new stack, in the order: GS, FS, DS, ES. These are pushed as 32-bit quantities, with undefined values in the upper 16 bits. Then load these 4 registers with null selectors (0).

(4) Push the old 8086 stack pointer onto the new stack by pushing the SS register (as 32-bits, high bits undefined), then pushing the 32-bit ESP register saved above.

(5) Push the 32-bit FLAGS register saved in step 1.

(6) Push the old 8086 instruction pointer onto the new stack by pushing the CS register (as 32-bits, high bits undefined), then pushing the 32-bit EIP register.

(7) Load up the new CS:EIP value from the interrupt gate, and begin execution of the interrupt routine in protected Intel486 DX Microprocessor mode.

The transition out of virtual 8086 mode performs a level change and stack switch, in addition to changing back to protected mode. In addition, all of the 8086 segment register images are stored on the stack (behind the SS:ESP image), and then loaded with null (0) selectors before entering the interrupt handler. This will permit the handler to safely save and restore the DS, ES, FS, and GS registers as 80286 selectors. This is needed so that interrupt handlers which don't care about the mode of the interrupted program can use the same prolog and epilog code for state saving (i.e., push all registers in prolog, pop all in epilog) regardless of whether or not a "native" mode or Virtual 8086 mode program was interrupted. Restoring null selectors to these registers before executing the IRET will not cause a trap in the interrupt handler. Interrupt routines which expect values in the segment registers, or return values in segment registers will have to obtain/return values from the 8086 register images pushed onto the new stack. They will need to know the mode of the interrupted program in order to know where to find/return segment registers, and also to know how to interpret segment register values.

The IRET instruction will perform the inverse of the above sequence. Only the extended Intel486 DX IRET instruction (operand size=32) can be used, and must be executed at level 0 to change the VM bit to 1.

(1) If the NT bit in the FLAGs register is on, an inter-task return is performed. The current state is stored in the current TSS, and the link field in the current TSS is used to locate the TSS for the interrupted task which is to be resumed.

Otherwise, continue with the following sequence.

(2) Read the FLAGS image from SS:8[ESP] into the FLAGS register. This will set VM to the value active in the interrupted routine.

(3) Pop off the instruction pointer CS:EIP. EIP is popped first, then a 32-bit word is popped which contains the CS value in the lower 16 bits. If VM=0, this CS load is done as a protected mode segment load. If VM=1, this will be done as an 8086 segment load.

(4) Increment the ESP register by 4 to bypass the FLAGS image which was "popped" in step 1.

(5) If VM=1, load segment registers ES, DS, FS, and GS from memory locations SS:[ESP+8], SS:[ESP+12], SS:[ESP+16], and SS:[ESP+20], respectively, where the new value of ESP stored in step 4 is used. Since VM=1, these are done as 8086 segment register loads.

Else if VM=0, check that the selectors in ES, DS, FS, and GS are valid in the interrupted routine. Null out invalid selectors to trap if an attempt is made to access through them.

(6) If (RPL(CS) > CPL), pop the stack pointer SS:ESP from the stack. The ESP register is popped first, followed by 32-bits containing SS in the lower 16 bits. If VM=0, SS is loaded as a protected mode segment register load. If VM=1, an 8086 segment register load is used.

(7) Resume execution of the interrupted routine. The VM bit in the FLAGS register (restored from the interrupt routine's stack image in step 1) determines whether the processor resumes the interrupted routine in Protected mode of Virtual 8086 mode.

5.0 ON-CHIP CACHE

To meet its performance goals the Intel486 DX2 microprocessor contains an eight Kbyte cache. The cache is software transparent to maintain binary compatibility with previous generations of the Intel386™/Intel486™ architecture.

The on-chip cache has been designed for maximum flexibility and performance. The cache has several operating modes offering flexibility during program execution and debugging. Memory areas can be defined as non-cacheable by software and external hardware. Protocols for cache line invalidations and replacement are implemented in hardware, easing system design.

5.1 Cache Organization

The on-chip cache is a unified code and data cache. The cache is used for both instruction and data accesses and acts on physical addresses.

The cache organization is 4-way set associative and each line is 16 bytes wide. The eight Kbytes of cache memory are logically organized as 128 sets, each containing four lines.

The cache memory is physically split into four 2-Kbyte blocks each containing 128 lines (see Figure 5.1). Associated with each 2-Kbyte block are 128 21-bit tags. There is a valid bit for each line in the cache. Each line in the cache is either valid or not valid. There are no provisions for partially valid lines.

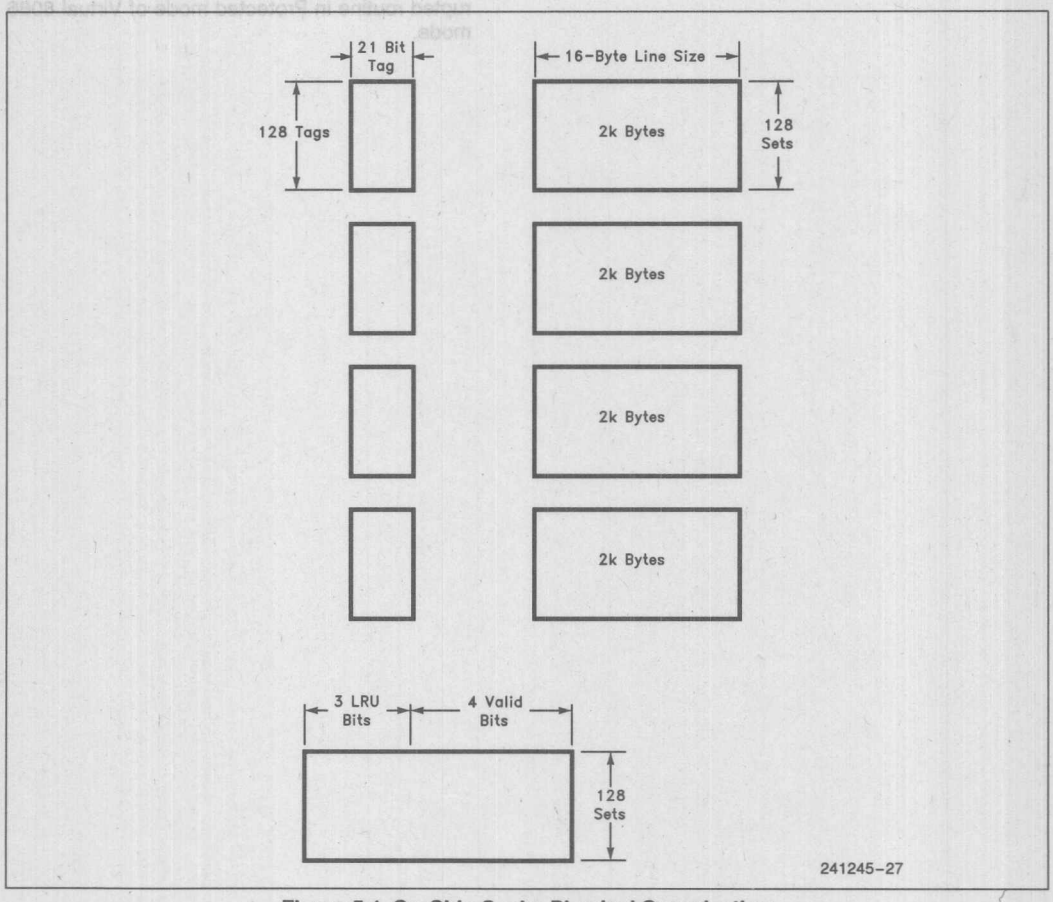


Figure 5.1. On-Chip Cache Physical Organization

The write strategy of on-chip cache is write-through. All writes will drive an external write bus cycle in addition to writing the information to the internal cache if the write was a cache hit. A write to an address not contained in the internal cache will only be written to external memory. Cache allocations are not made on write misses.

5.2 Cache Control

Control of the cache is provided by the CD and NW bits in CR0. CD enables and disables the cache. NW controls memory write-through and invalidates.

The CD and NW bits define four operating modes of the on-chip cache as given in Table 5.1. These modes provide flexibility in how the on-chip cache is used.

Table 5.1. Cache Operating Modes

CD	NW	Operating Mode
1	1	Cache fills disabled, write-through and invalidates disabled
1	0	Cache fills disabled, write-through and invalidates enabled
0	1	INVALID. IF CR0 is loaded with this configuration of bits, a GP fault with error code of 0 is raised.
0	0	Cache fills enabled, write-through and invalidates enabled

CD = 1, NW = 1

The cache is completely disabled by setting CD = 1 and NW = 1 and then flushing the cache. This mode may be useful for debugging programs where it is important to see all memory cycles at the pins. Writes which hit in the cache will not appear on the external bus.

It is possible to use the on-chip cache as fast static RAM by "pre-loading" certain memory areas into the cache and then setting CD = 1 and NW = 1. Pre-loading can be done by careful choice of memory references with the cache turned on or by use of the testability functions (see Section 8.2). When the cache is turned off the memory mapped by the cache is "frozen" into the cache since fills and invalidates are disabled.

CD = 1, NW = 0

Cache fills are disabled but write-throughs and invalidates are enabled. This mode is the same as if the KEN# pin was strapped HIGH disabling cache fills. Write-throughs and invalidates may still occur to keep the cache valid. This mode is useful if the software must disable the cache for a short period of time, and then re-enable it without flushing the original contents.

CD = 0, NW = 1

INVALID. If CR0 is loaded with this bit configuration, a General Protection fault with error code of 0 is raised. Note that this mode would imply a non-transparent write-back cache. A future processor may define this combination of bits to implement a write-back cache.

CD = 0, NW = 0

This is the normal operating mode.

Completely disabling the cache is a two step process. First CD and NW must be set to 1 and then the cache must be flushed. If the cache is not flushed, cache hits on reads will still occur and data will be read from the cache.

5.3 Cache Line Fills

Any area of memory can be cached in the Intel486 DX microprocessor. Non-cacheable portions of memory can be defined by the external system or by software. The external system can inform the Intel486 DX microprocessor that a memory address is non-cacheable by returning the KEN# pin inactive during a memory access (refer to Section 7.2.3). Software can prevent certain pages from being cached by setting the PCD bit in the page table entry.

A read request can be generated from program operation or by an instruction pre-fetch. The data will be supplied from the on-chip cache if a cache hit occurs on the read address. If the address is not in the cache, a read request for the data is generated on the external bus.

If the read request is to a cacheable portion of memory, the Intel486 DX microprocessor initiates a cache line fill. During a line fill a 16-byte line is read into the Intel486 DX microprocessor.

Cache fills will only be generated for read misses. Write misses will never cause a line in the internal cache to be allocated. If a cache hit occurs on a write, the line will be updated.

Cache line fills can be performed over 8- and 16-bit busses using the dynamic bus sizing feature. Refer to Section 7.1.3 for a description of dynamic bus sizing.

Refer to Section 7.2.3 for further information on cacheable cycles.

5.4 Cache Line Invalidations

The Intel486 DX microprocessor contains both a hardware and software mechanism for invalidating lines in its internal cache. Cache line invalidations are needed to keep the Intel486 DX microprocessor's cache contents consistent with external memory.

Refer to Section 7.2.8 for further information on cache line invalidations.

5.5 Cache Replacement

When a line needs to be placed in its internal cache the Intel486 DX microprocessor first checks to see if there is a non-valid line in the set that can be replaced. If all four lines in the set are valid, a pseudo least-recently-used mechanism is used to determine which line should be replaced.

A valid bit is associated with each line in the cache. When a line needs to be placed in a set, the four valid bits are checked to see if there is a non-valid line that can be replaced. If a non-valid line is found, that line is marked for replacement.

The four lines in the set are labeled I0, I1, I2, and I3. The order in which the valid bits are checked during an invalidation is I0, I1, I2 and I3. All valid bits are cleared when the processor is reset or when the cache is flushed.

Replacement in the cache is handled by a pseudo least recently used (LRU) mechanism when all four lines in a set are valid. Three bits, B0, B1 and B2, are defined for each of the 128 sets in the cache. These bits are called the LRU bits. The LRU bits are updated for every hit or replace in the cache.

If the most recent access to the set was to I0 or I1, B0 is set to 1. B0 is set to 0 if the most recent access was to I2 or I3. If the most recent access to I0:I1 was to I0, B1 is set to 1, else B1 is set to 0. If the most recent access to I2:I3 was to I2, B2 is set to 1, else B2 is set to 0.

The pseudo LRU mechanism works in the following manner. When a line must be replaced, the cache will first select which of I0:I1 and I2:I3 was least recently used. Then the cache will determine which of the two lines was least recently used and mark it for replacement. This decision tree is shown in Figure 5.2. When the processor is reset or when the cache is flushed all 128 sets of three LRU bits are set to 0.

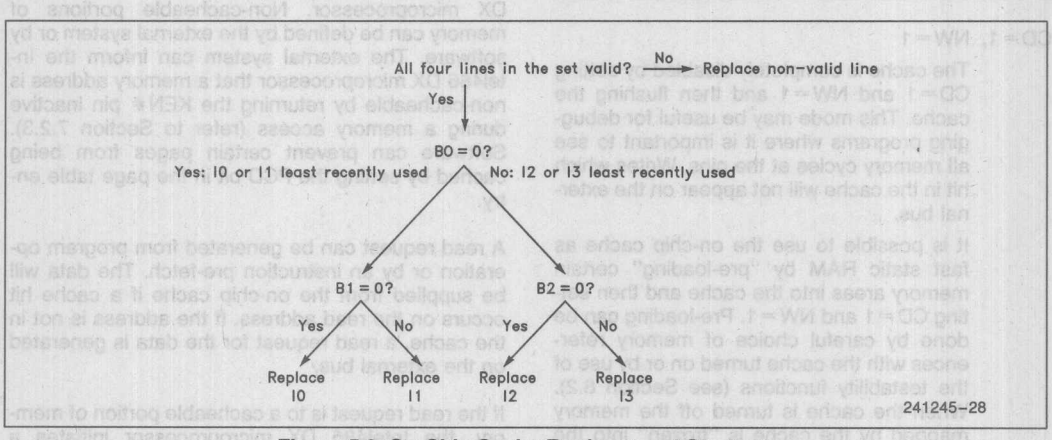


Figure 5.2. On-Chip Cache Replacement Strategy

5.6 Page Cacheability

Two bits for cache control, PWT and PCD, are defined in the page table and page directory entries. The state of these bits are driven out on the PWT and PCD pins during memory access cycles.

The PWT bit controls write policy for second level caches used with the Intel486 DX microprocessor. Setting PWT = 1 defines a write-through policy for the current page while PWT = 0 allows the possibility of write-back. The state of PWT is ignored internally by the Intel486 DX microprocessor since the on-chip cache is write through.

The PCD bit controls cacheability on a page by page basis. The PCD bit is internally ANDed with the KEN# signal to control cacheability on a cycle by cycle basis (see Figure 5.3). PCD = 0 enables caching while PCD = 1 forbids it. Note that cache fills are enabled when PCD = 0 AND KEN# = 0. This logical AND is implemented physically with a NOR gate.

The state of the PCD bit in the page table entry is driven on the PCD pin when a page in external memory is accessed. The state of the PCD pin informs the external system of the cacheability of the requested information. The external system then returns KEN# telling the Intel486 DX microprocessor if the area is cacheable. The Intel486 DX microprocessor initiates a cache line fill if PCD and KEN# indicate that the requested information is cacheable.

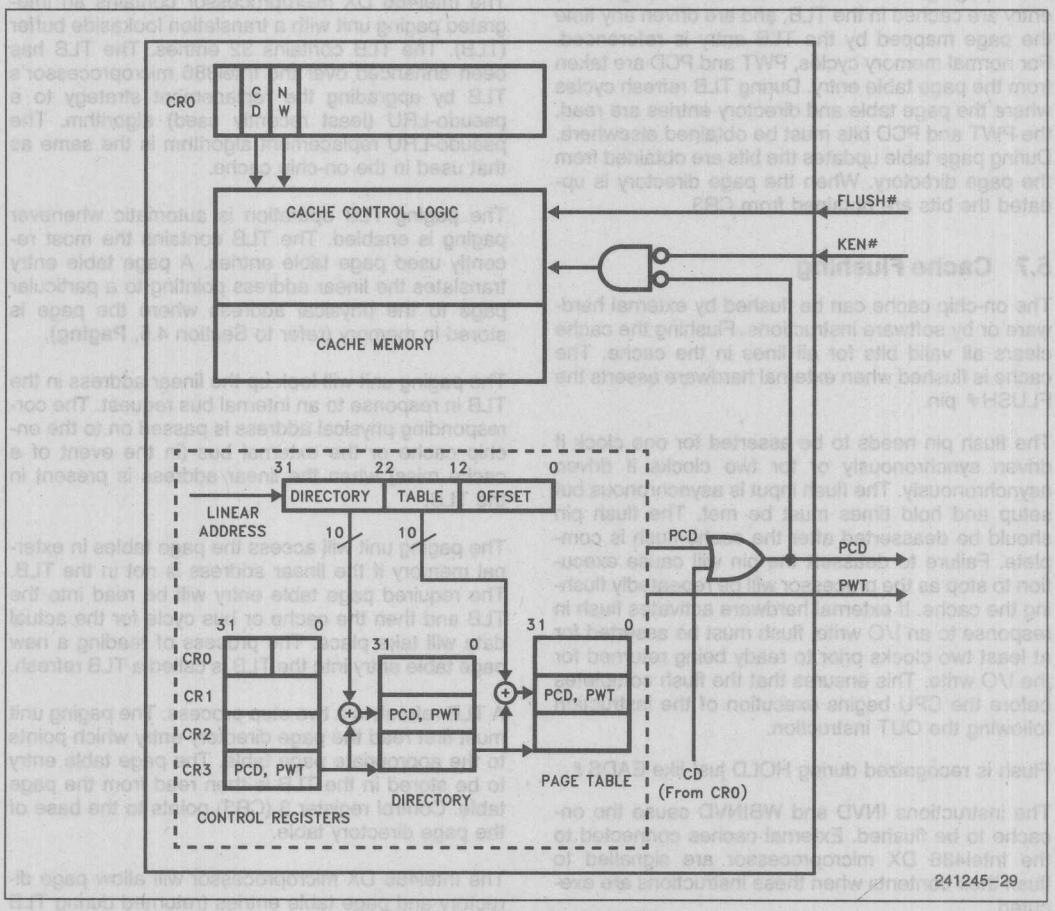


Figure 5.3. Page Cacheability

The PCD bit is masked with the CD (cache disable) bit in control register 0 to determine the state of the PCD pin. If CD=1 the Intel486 DX microprocessor forces the PCD pin HIGH. If CD=0 the PCD pin is driven with the value for the page table entry/directory. See Figure 5.3.

The PWT and PCD bits for a bus cycle are obtained from either CR3, the page directory or page table entry. These bits are assumed to be zero during real mode, whenever paging is disabled, or for cycles that bypass paging, (I/O references, interrupt acknowledge and Halt cycles), the PWT and PCD bits are taken from CR3. These bits are initialized to 0 on reset, but can be set to any value by level 0 software.

When paging is enabled, the bits from the page table entry are cached in the TLB, and are driven any time the page mapped by the TLB entry is referenced. For normal memory cycles, PWT and PCD are taken from the page table entry. During TLB refresh cycles where the page table and directory entries are read, the PWT and PCD bits must be obtained elsewhere. During page table updates the bits are obtained from the page directory. When the page directory is updated the bits are obtained from CR3.

5.7 Cache Flushing

The on-chip cache can be flushed by external hardware or by software instructions. Flushing the cache clears all valid bits for all lines in the cache. The cache is flushed when external hardware asserts the FLUSH# pin.

The flush pin needs to be asserted for one clock if driven synchronously or for two clocks if driven asynchronously. The flush input is asynchronous but setup and hold times must be met. The flush pin should be deasserted after the cache flush is complete. Failure to deassert the pin will cause execution to stop as the processor will be repeatedly flushing the cache. If external hardware activates flush in response to an I/O write, flush must be asserted for at least two clocks prior to ready being returned for the I/O write. This ensures that the flush completes before the CPU begins execution of the instruction following the OUT instruction.

Flush is recognized during HOLD just like EADS#.

The instructions INVD and WBINVD cause the on-cache to be flushed. External caches connected to the Intel486 DX microprocessor are signalled to flush their contents when these instructions are executed.

WBINVD will cause an external write-back cache to write back dirty lines before flushing its contents. The external cache is signalled using the bus cycle definition pins and the byte enables (refer to Section 6.2.5 for the bus cycle definition pins and Section 7.2.11 for special bus cycles). Refer to the Intel486 DX microprocessor programmers reference manual for detailed instruction definitions.

The results of the INVD and WBINVD instructions are identical for the operation of the Intel486 DX microprocessor's on-chip cache since the cache is write-through. Note that the INVD and WBINVD instructions are machine dependent. Future members of the Intel486 DX microprocessor family may change the definition of this instruction.

5.8 Caching Translation Lookaside Buffer Entries

The Intel486 DX microprocessor contains an integrated paging unit with a translation lookaside buffer (TLB). The TLB contains 32 entries. The TLB has been enhanced over the Intel386 microprocessor's TLB by upgrading the replacement strategy to a pseudo-LRU (least recently used) algorithm. The pseudo-LRU replacement algorithm is the same as that used in the on-chip cache.

The paging TLB operation is automatic whenever paging is enabled. The TLB contains the most recently used page table entries. A page table entry translates the linear address pointing to a particular page to the physical address where the page is stored in memory (refer to Section 4.5, **Paging**).

The paging unit will look up the linear address in the TLB in response to an internal bus request. The corresponding physical address is passed on to the on-chip cache or the external bus (in the event of a cache miss) when the linear address is present in the TLB.

The paging unit will access the page tables in external memory if the linear address is not in the TLB. The required page table entry will be read into the TLB and then the cache or bus cycle for the actual data will take place. The process of reading a new page table entry into the TLB is called a TLB refresh.

A TLB refresh is a two step process. The paging unit must first read the page directory entry which points to the appropriate page table. The page table entry to be stored in the TLB is then read from the page table. Control register 3 (CR3) points to the base of the page directory table.

The Intel486 DX microprocessor will allow page directory and page table entries (returned during TLB refreshes) to be stored in the on-chip cache. Setting the PCD bits in CR3 and the page directory entry to 1 will prevent the page directory and page table entries from being stored in the on-chip cache (see Section 5.6, **Page Cacheability**).

6.0 HARDWARE INTERFACE

6.1 Introduction

The Intel486 DX2 microprocessor bus has been designed to be identical with the Intel486 microprocessor bus. Several new features have been added to the Intel486 DX2 to increase performance and testability. New features include a speed doubler for the core logic, and IEEE 1149.1 boundary scan support.

The Intel486 DX2 is driven by what can be called a ½x clock, as opposed to the 1x clock in the Intel486 DX and the 2x clock in the Intel386 microprocessors. Thus a 50 MHz Intel486 DX2 is driven by a 25 MHz clock, in contrast with 25 MHz processors like the Intel486 DX2 and the Intel386 requiring 25 MHz and 50 MHz, respectively. Since the Intel486 DX has a clock doubler driving its core, but not its bus interface, it provides a simpler system design for a given performance level than the Intel486 DX. In reality, this permits dramatic increases in performance by just plugging in the Intel486 DX2 in a system that had already been designed for the Intel486. This performance is supplied because the bus interface is identical to that of an Intel486 DX and all of the core is running at twice the speed of the comparable Intel486 DX. This speedup includes the internal cache memory, the floating point unit, the instruction decode unit, the ALU and everything except the bus interface.

Like the Intel386 microprocessor, the Intel486 DX microprocessor has separate parallel busses for data and addresses. The bidirectonal data bus is 32 bits in width. The address bus consists of two components: 30 address lines (A2–A31) and 4 byte enable lines (BE0#–BE3#). The address bus addresses external memory in the same manner as the Intel386 microprocessor: The address lines form the upper 30 bits of the address and the byte enables select individual bytes within a 4 byte location. The address lines are bidirectional for use in cache line invalidations.

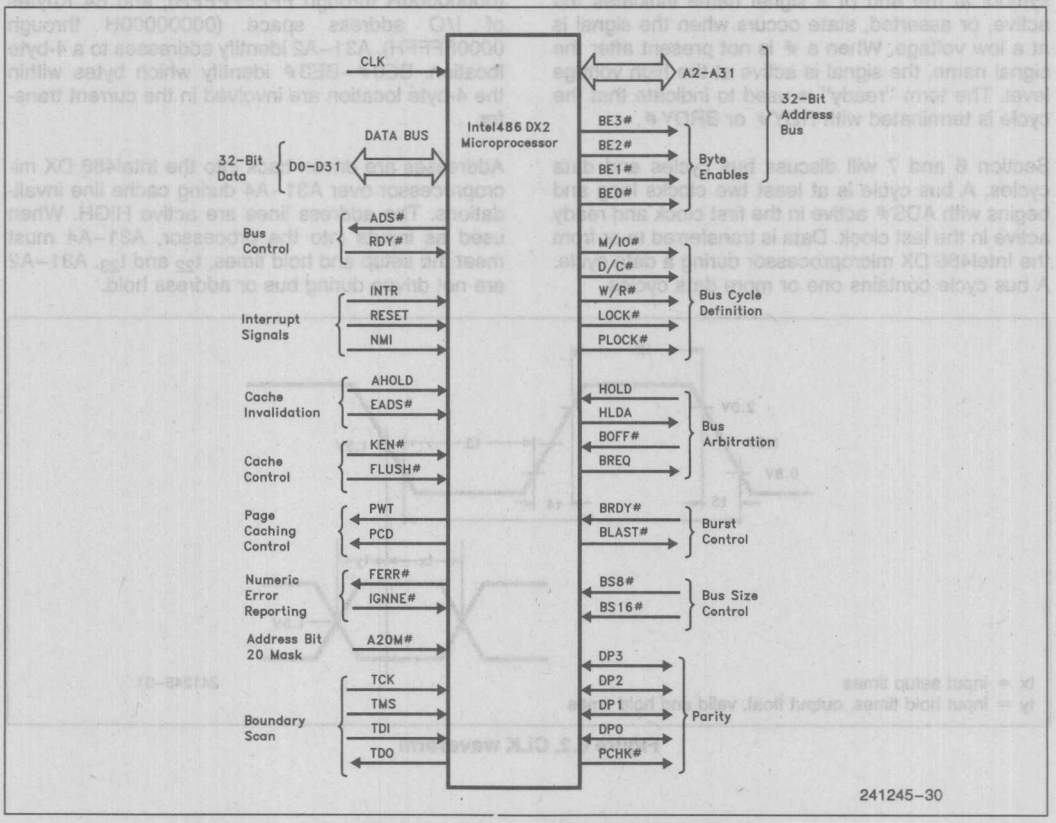


Figure 6.1. Functional Signal Groupings

The Intel486 DX microprocessor's burst bus mechanism enables high-speed cache fills from external memory. Burst cycles can strobe data into the processor at a rate of one item every clock. Non-burst cycles have a maximum rate of one item every two clocks. Burst cycles are not limited to cache fills: all bus cycles requiring more than a single data cycle can be bursted.

The Intel486 DX microprocessor has a bus hold feature similar to that of the Intel386 microprocessor. During bus hold, the Intel486 DX microprocessor relinquishes control of the local bus by floating its address, data and control busses.

The Intel486 DX microprocessor has an address hold feature in addition to bus hold. During address hold only the address bus is floated, the data and control busses can remain active. Address hold is used for cache line invalidations.

Ahead is a brief description of the Intel486 DX microprocessor input and output signals arranged by functional groups. Before beginning the signal descriptions a few terms need to be defined. The # symbol at the end of a signal name indicates the active, or asserted, state occurs when the signal is at a low voltage. When a # is not present after the signal name, the signal is active at the high voltage level. The term "ready" is used to indicate that the cycle is terminated with RDY# or BRDY#.

Section 6 and 7 will discuss bus cycles and data cycles. A bus cycle is at least two clocks long and begins with ADS# active in the first clock and ready active in the last clock. Data is transferred to or from the Intel486 DX microprocessor during a data cycle. A bus cycle contains one or more data cycles.

6.2 Signal Descriptions

6.2.1 CLOCK (CLK)

CLK provides the fundamental timing and the internal operating frequency for the Intel486 DX2 microprocessor. All external timing parameters are specified with respect to the **rising edge** of CLK.

The Intel486 DX2 microprocessor can operate over a wide frequency range but CLK's frequency cannot change rapidly while RESET is inactive. CLK's frequency must be stable for proper chip operation since a single edge of CLK is used internally to generate four phases. CLK only needs TTL levels for proper operation. Figure 6.2 illustrates the CLK waveform.

6.2.2 Address Bus (A31–A2, BE0#–BE3#)

A31–A2 and BE0#–BE3# form the address bus and provide physical memory and I/O port addresses. The Intel486 DX microprocessor is capable of addressing 4 gigabytes of physical memory space (00000000H through FFFFFFFFH), and 64 Kbytes of I/O address space (00000000H through 0000FFFFH). A31–A2 identify addresses to a 4-byte location. BE0#–BE3# identify which bytes within the 4-byte location are involved in the current transfer.

Addresses are driven back into the Intel486 DX microprocessor over A31–A4 during cache line invalidations. The address lines are active HIGH. When used as inputs into the processor, A31–A4 must meet the setup and hold times, t_{22} and t_{23}. A31–A2 are not driven during bus or address hold.

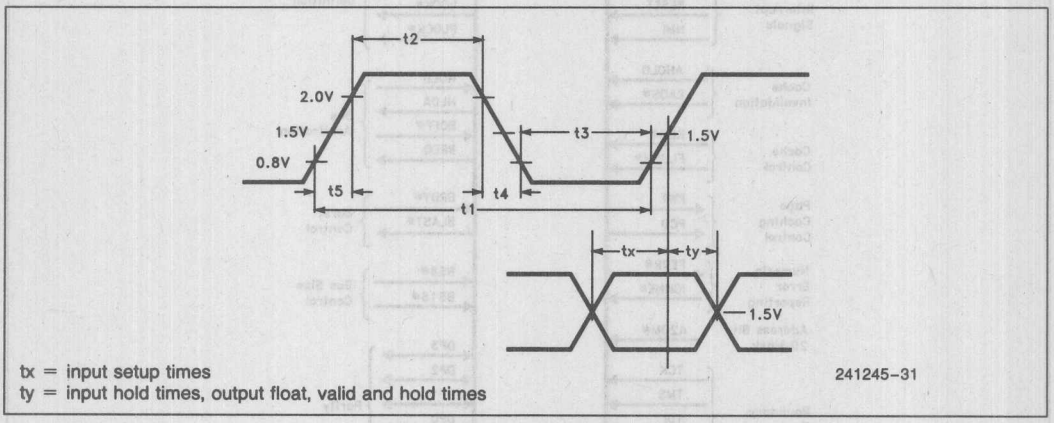


Figure 6.2. CLK waveform

The byte enable outputs, BE0#–BE3#, determine which bytes must be driven valid for read and write cycles to external memory.

BE3# applies to D24–D31

BE2# applies to D16–D23

BE1# applies to D8–D15

BE0# applies to D0–D7

BE0#–BE3# can be decoded to generate A0, A1 and BHE# signals used in 8- and 16-bit systems (see Table 7.5). BE0#–BE3# are active LOW and are not driven during bus hold.

6.2.3 DATA LINES (D31–D0)

The bidirectional lines, D31–D0, form the data bus for the Intel486 DX microprocessor. D0–D7 define the least significant byte and D24–D31 the most significant byte. Data transfers to 8- or 16-bit devices is possible using the data bus sizing feature controlled by the BS8# or BS16# input pins.

D31–D0 are active HIGH. For reads, D31–D0 must meet the setup and hold times, t_{22} and t_{23}. D31–D0 are not driven during read cycles and bus hold.

6.2.4 PARITY

Data Parity Input/Outputs (DP0–DP3)

DP0–DP3 are the data parity pins for the processor. There is one pin for each byte of the data bus. Even parity is generated or checked by the parity generators/checkers. Even parity means that there are an even number of HIGH inputs on the eight corresponding data bus pins and parity pin.

Data parity is generated on all write data cycles with the same timing as the data driven by the Intel486 DX microprocessor. Even parity information must be driven back to the Intel486 DX microprocessor on these pins with the same timing as read information to insure that the correct parity check status is indicated by the Intel486 DX microprocessor.

The values read on these pins do not affect program execution. It is the responsibility of the system to take appropriate actions if a parity error occurs.

Input signals on DP0–DP3 must meet setup and hold times t_{22} and t_{23} for proper operation.

Parity Status Output (PCHK#)

Parity status is driven on the PCHK# pin, and a parity error is indicated by this pin being LOW. PCHK# is driven the clock after ready for read operations to indicate the parity status for the data sampled at the end of the previous clock. Parity is checked during code reads, memory reads and I/O reads. Parity is not checked during interrupt acknowledge cycles. PCHK# only checks the parity status for enabled bytes as indicated by the byte enable and bus size signals. It is valid only in the clock immediately after read data is returned to the Intel486 DX microprocessor. At all other times it is inactive (HIGH). PCHK# is never floated.

Driving PCHK# is the only effect that bad input parity has on the Intel486 DX microprocessor. The Intel486 DX microprocessor will not vector to a bus error interrupt when bad data parity is returned. In systems that will not employ parity, PCHK# can be ignored. In systems not using parity, DP0–DP3 should be connected to V_{CC} through a pullup resistor.

2

6.2.5 BUS CYCLE DEFINITION

M/IO#, D/C#, W/R# Outputs

M/IO#, D/C# and W/R# are the primary bus cycle definition signals. They are driven valid as the ADS# signal is asserted. M/IO# distinguishes between memory and I/O cycles, D/C# distinguishes between data and control cycles and W/R# distinguishes between write and read cycles.

Bus cycle definitions as a function of M/IO#, D/C# and W/R# are given in Table 6.1. Note there is a difference between the Intel486 DX microprocessor and Intel386 microprocessor bus cycle definitions. The halt bus cycle type has been moved to location 001 in the Intel486 DX microprocessor from location 101 in the Intel386 microprocessor. Location 101 is now reserved and will never be generated by the Intel486 DX microprocessor.

Table 6.1. ADS# Initiated Bus Cycle Definitions

M/IO#	D/C#	W/R#	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Halt/Special Cycle
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Code Read
1	0	1	Reserved
1	1	0	Memory Read
1	1	1	Memory Write

Special bus cycles are discussed in Section 7.2.11.

Bus Lock Output (LOCK#)

LOCK# indicates that the Intel486 DX microprocessor is running a read-modify-write cycle where the external bus must not be relinquished between the

read and write cycles. Read-modify-write cycles are used to implement memory-based semaphores. Multiple reads or writes can be locked.

When LOCK# is asserted, the current bus cycle is locked and the Intel486 DX microprocessor should be allowed exclusive access to the system bus. LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after ready is returned indicating the last locked bus cycle.

The Intel486 DX microprocessor will not acknowledge bus hold when LOCK# is asserted (though it will allow an address hold). LOCK# is active LOW and is floated during bus hold. Locked read cycles will not be transformed into cache fill cycles if KEN# is returned active. Refer to Section 7.2.6 for a detailed discussion of Locked bus cycles.

Pseudo-Lock Output (PLOCK#)

The pseudo-lock feature allows atomic reads and writes of memory operands greater than 32 bits. These operands require more than one cycle to transfer. The Intel486 DX microprocessor asserts PLOCK# during floating point long reads and writes (64 bits), segment table descriptor reads (64 bits) and cache line fills (128 bits).

When PLOCK# is asserted no other master will be given control of the bus between cycles. A bus hold request (HOLD) is not acknowledged during pseudo-locked reads and writes, with one exception. During non-cacheable non-bursted code prefetches, HOLD is recognized on memory cycle boundaries even though PLOCK# is asserted. The Intel486 DX microprocessor will drive PLOCK# active until the addresses for the last bus cycle of the transaction have been driven regardless of whether BRDY# or RDY# are returned.

A pseudo-locked transfer is meaningful only if the memory operand is aligned and if its completely contained within a single cache line. A 64-bit floating point number must be aligned to an 8-byte boundary to guarantee an atomic access.

Normally PLOCK# and BLAST# are inverse of each other. However during the first cycle of a 64-bit floating point write, both PLOCK# and BLAST# will be asserted.

Since PLOCK# is a function of the bus size and KEN# inputs, PLOCK# should be sampled only in the clock ready is returned. This pin is active LOW and is not driven during bus hold. Refer to Section 7.2.7 for a detailed discussion of pseudo-locked bus cycles.

6.2.6 BUS CONTROL

The bus control signals allow the processor to indicate when a bus cycle has begun, and allow other system hardware to control burst cycles, data bus width and bus cycle termination.

Address Status Output (ADS#)

The ADS# output indicates that the address and bus cycle definition signals are valid. This signal will go active in the first clock of a bus cycle and go inactive in the second and subsequent clocks of the cycle. ADS# is also inactive when the bus is idle.

ADS# is used by external bus circuitry as the indication that the processor has started a bus cycle. The external circuit must sample the bus cycle definition pins on the next rising edge of the clock after ADS# is driven active.

ADS# is active LOW and is not driven during bus hold.

Non-burst Ready Input (RDY#)

RDY# indicates that the current bus cycle is complete. In response to a read, RDY# indicates that the external system has presented valid data on the data pins. In response to a write request, RDY# indicates that the external system has accepted the Intel486 DX microprocessor data. RDY# is ignored when the bus is idle and at the end of the first clock of the bus cycle. Since RDY# is sampled during address hold, data can be returned to the processor when AHOLD is active.

RDY# is active LOW, and is not provided with an internal pullup resistor. This input must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.

6.2.7 BURST CONTROL

Burst Ready Input (BRDY#)

BRDY# performs the same function during a burst cycle that RDY# performs during a non-burst cycle. BRDY# indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted the Intel486 DX microprocessor data in response to a write. BRDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle.

During a burst cycle, BRDY# will be sampled each clock, and if active, the data presented on the data bus pins will be strobed into the Intel486 DX microprocessor. ADS# is negated during the second through last data cycles in the burst, but address

lines A2–A3 and byte enables will change to reflect the next data item expected by the Intel486 DX microprocessor.

If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is prematurely aborted. An additional complete bus cycle will be initiated after an aborted burst cycle if the cache line fill was not complete. BRDY# is treated as a normal ready for the last data cycle in a burst transfer or for non-burstable cycles. Refer to Section 7.2.2 for burst cycle timing.

BRDY# is active LOW and is provided with a small internal pullup resistor. BRDY# must satisfy the setup and hold times t_{16} and t_{17}.

Burst Last Output (BLAST#)

BLAST# indicates that the next time BRDY# is returned it will be treated as a normal RDY#, terminating the line fill or other multiple-data-cycle transfer. BLAST# is active for all bus cycles regardless of whether they are cacheable or not. This pin is active LOW and is not driven during bus hold.

6.2.8 INTERRUPT SIGNALS (RESET, INTR, NMI)

The interrupt signals can interrupt or suspend execution of the processor's current instruction stream.

Reset Input (RESET)

RESET forces the Intel486 DX2 microprocessor to begin execution at a known state. **For a power-up (cold start) reset, V_{CC} and CLK must reach their proper DC and AC specifications for at least 1 ms** before the Intel486 DX2 microprocessor begins instruction execution. The RESET pin should remain active during this time to ensure proper Intel486 DX2 microprocessor operation. However, **for a warm boot-up case, RESET is required to remain active for a minimum of 15 clocks.** The testability operating modes are programmed by the falling (inactive going) edge of RESET. (Refer to Section 8.0 for a description of the test modes during reset.)

Maskable Interrupt Request Input (INTR)

INTR indicates that an external interrupt has been generated. Interrupt processing is initiated if the IF flag is active in the EFLAGS register.

The Intel486 DX microprocessor will generate two locked interrupt acknowledge bus cycles in response to asserting the INTR pin. An 8-bit interrupt number will be latched from an external interrupt controller at the end of the second interrupt acknowledge cycle. INTR must remain active until the interrupt acknowledges have been performed to assure program interruption. Refer to Section 7.2.10 for a detailed discussion of interrupt acknowledge cycles.

The INTR pin is active HIGH and is not provided with an internal pulldown resistor. INTR is asynchronous, but the INTR setup and hold times, t_{20} and t_{21}, must be met to assure recognition on any specific clock.

Non-maskable Interrupt Request Input (NMI)

NMI is the non-maskable interrupt request signal. Asserting NMI causes an interrupt with an internally supplied vector value of 2. External interrupt acknowledge cycles are not generated since the NMI interrupt vector is internally generated. When NMI processing begins, the NMI signal will be masked internally until the IRET instruction is executed.

NMI is rising edge sensitive after internal synchronization. NMI must be held LOW for at least four CLK periods before this rising edge for proper operation. NMI is not provided with an internal pulldown resistor. NMI is asynchronous but setup and hold times, t_{20} and t_{21} must be met to assure recognition on any specific clock.

6.2.9 BUS ARBITRATION SIGNALS

This section describes the mechanism by which the processor relinquishes control of its local bus when requested by another bus master.

Bus Request Output (BREQ)

The Intel486 DX microprocessor asserts BREQ whenever a bus cycle is pending internally. Thus, BREQ is always asserted in the first clock of a bus cycle, along with ADS#. Furthermore, if the Intel486 DX microprocessor is currently not driving the bus (due to HOLD, AHOLD, or BOFF#), BREQ is asserted in the same clock that ADS# would have been asserted if the processor were driving the bus. After the first clock of the bus cycle, BREQ may change state. It will be asserted if additional cycles are necessary to complete a transfer (via BS8#, BS16#, KEN#), or if more cycles are pending internally. However, if no additional cycles are necessary to complete the current transfer, BREQ can be negated before ready comes back for the current cycle. External logic can use the BREQ signal to arbitrate among multiple processors. This pin is driven regardless of the state of bus hold or address hold. BREQ is active HIGH and is never floated. During a hold state, internal events may cause BREQ to be deasserted prior to any bus cycles.

Bus Hold Request Input (HOLD)

HOLD allows another bus master complete control of the Intel486 DX microprocessor bus. The Intel486

DX microprocessor will respond to an active HOLD signal by asserting HLDA and placing most of its output and input/output pins in a high impedance state (floated) after completing its current bus cycle, burst cycle, or sequence of locked cycles. In addition, if the Intel486 DX CPU receives a HOLD request while performing a code fetch, and that cycle is backed off (BOFF#), the Intel486 DX CPU will recognize HOLD before restarting the cycle. The BREQ, HLDA, PCHK# and FERR# pins are not floated during bus hold. The Intel486 DX microprocessor will maintain its bus in this state until the HOLD is deasserted. Refer to Section 7.2.9 for timing diagrams for bus hold cycles and HOLD request acknowledge during BOFF#.

Unlike the Intel386 microprocessor, the **Intel486 DX microprocessor will recognize HOLD during reset.** Pullup resistors are not provided for the outputs that are floated in response to HOLD. HOLD is active HIGH and is not provided with an internal pulldown resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper chip operation.

Bus Hold Acknowledge Output (HLDA)

HLDA indicates that the Intel486 DX microprocessor has given the bus to another local bus master. HLDA goes active in response to a hold request presented on the HOLD pin. HLDA is driven active in the same bus clock cycle that the Intel486 DX microprocessor floats its bus.

HLDA will be driven inactive when leaving bus hold and the Intel486 DX microprocessor will resume driving the bus. The Intel486 DX microprocessor will not cease internal activity during bus hold since the internal cache will satisfy the majority of bus requests. HLDA is active HIGH and remains driven during bus hold.

Backoff Input (BOFF#)

Asserting the BOFF# input forces the Intel486 DX microprocessor to release control of its bus in the next clock. The pins floated are exactly the same as in response to HOLD. The response to BOFF# differs from the response to HOLD in two ways: First, the bus is floated immediately in response to BOFF# while the Intel486 DX microprocessor completes the current bus cycle before floating its bus in response to HOLD. Second the Intel486 DX does not assert HLDA in response to BOFF#.

The processor remains in bus hold until BOFF# is negated. Upon negation, the Intel486 DX microprocessor restarts the bus cycle aborted when BOFF# was asserted. To the internal execution engine the effect of BOFF# is the same as inserting a few wait states to the original cycle. Refer to Section 7.2.12 for a description of bus cycle restart.

Any data returned to the processor while BOFF# is asserted is ignored. BOFF# has higher priority than RDY# or BRDY#. If both BOFF# and ready are returned in the same clock, BOFF# takes effect. If BOFF# is asserted while the bus is idle, the Intel486 DX microprocessor will float its bus in the next bus clock. BOFF# is active LOW and must meet setup and hold times t_{18} and t_{19} for proper chip operation.

6.2.10 CACHE INVALIDATION

The AHOLD and EADS# inputs are used during cache invalidation cycles. AHOLD conditions the Intel486 DX microprocessors address lines, A4–A31, to accept an address input. EADS# indicates that an external address is actually valid on the address inputs. Activating EADS# will cause the Intel486 DX microprocessor to read the external address bus and perform an internal cache invalidation cycle to the address indicated. Refer to Section 7.2.8 for cache invalidation cycle timing.

Address Hold Request Input (AHOLD)

AHOLD is the address hold request. It allows another bus master access to the Intel486 DX microprocessor address bus for performing an internal cache invalidation cycle. Asserting AHOLD will force the Intel486 DX microprocessor to stop driving its address bus in the next clock. While AHOLD is active only the address bus will be floated, the remainder of the bus can remain active. For example, data can be returned for a previously specified bus cycle when AHOLD is active. The Intel486 DX microprocessor will not initiate another bus cycle during address hold. Since the Intel486 DX microprocessor floats its bus immediately in response to AHOLD, an address hold acknowledge is not required. If AHOLD is asserted while a bus cycle is in progress, and no readies are returned during the time AHOLD is asserted, the Intel486 DX will redrive the same address (that it originally sent out) once AHOLD is negated.

AHOLD is recognized during reset. Since the entire cache is invalidated by reset, any invalidation cycles run during reset will be unnecessary. AHOLD is active HIGH and is provided with a small internal pulldown resistor. It must satisfy the setup and hold times t_{18} and t_{19} for proper chip operation. This pin determines whether or not the built in self test features of the Intel486 DX microprocessor will be exercised on assertion of RESET.

External Address Valid Input (EADS#)

EADS# indicates that a valid external address has been driven onto the Intel486 DX address pins. This address will be used to perform an internal cache invalidation cycle. The external address will be checked with the current cache contents. If the ad-

dress specified matches any areas in the cache, that area will immediately be invalidated.

An invalidation cycle may be run by asserting EADS# regardless of the state of AHOLD, HOLD and BOFF#. EADS# is active LOW and is provided with an internal pullup resistor. EADS# must satisfy the setup and hold times t_{12} and t_{13} for proper chip operation.

6.2.11 CACHE CONTROL

Cache Enable Input (KEN#)

KEN# is the cache enable pin. KEN# is used to determine whether the data being returned by the current cycle is cacheable. When KEN# is active and the Intel486 DX microprocessor generates a cycle that can be cached (most any memory read cycle), the cycle will be transformed into a cache line fill cycle.

A cache line is 16 bytes long. During the first cycle of a cache line fill the byte-enable pins should be ignored and data should be returned as if all four byte enables were asserted. The Intel486 DX microprocessor will run between 4 and 16 contiguous bus cycles to fill the line depending on the bus data width selected by BS8# and BS16#. Refer to Section 7.2.3 for a description of cache line fill cycles.

The KEN# input is active LOW and is provided with a small internal pullup resistor. It must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

Cache Flush Input (FLUSH#)

The FLUSH# input forces the Intel486 DX microprocessor to flush its entire internal cache. FLUSH# is active LOW and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition on any specific clock.

FLUSH# also determines whether or not the 3-state test mode of the Intel486 DX microprocessor will be invoked on assertion of RESET.

6.2.12 PAGE CACHEABILITY (PWT, PCD)

The PWT and PCD output signals correspond to two user attribute bits in the page table entry. When paging is enabled, PWT and PCD correspond to bits 3 and 4 of the page table entry respectively. For cycles that are not paged when paging is enabled (for example I/O cycles) PWT and PCD correspond to bits 3 and 4 in control register 3. When paging is disabled, the Intel486 DX CPU ignores the PCD and PWT bits and assumes they are zero for the purpose of caching and driving PCD and PWT.

PCD is masked by the CD (cache disable) bit in control register 0 (CR0). When CD=1 (cache line fills disabled) the Intel486 DX microprocessor forces PCD HIGH. When CD=0, PCD is driven with the value of the page table entry/directory.

The purpose of PCD is to provide a cacheable/non-cacheable indication on a page by page basis. The Intel486 DX will not perform a cache fill to any page in which bit 4 of the page table entry is set. PWT corresponds to the write-back bit and can be used by an external cache to provide this functionality. PCD and PWT bits are assigned to be zero during real mode or whenever paging is disabled. Refer to Sections 4.5.4 and 5.6 for a discussion of non-cacheable pages.

PCD and PWT have the same timing as the cycle definition pins (M/IO#, D/C#, W/R#). PCD and PWT are active HIGH and are not driven during bus hold.

6.2.13 NUMERIC ERROR REPORTING (FERR#, IGNNE#)

To allow PC-type floating point error reporting, the Intel486 DX microprocessor provides two pins, FERR# and IGNNE#.

Floating Point Error Output (FERR#)

The Intel486 DX microprocessor asserts FERR# whenever an unmasked floating point error is encountered. FERR# is similar to the ERROR# pin on the Intel387 math coprocessor. FERR# can be used by external logic for PC-type floating point error reporting in Intel486 DX microprocessor systems. FERR# is active LOW, and is not floated during bus hold.

In some cases, FERR# is asserted when the next floating point instruction is encountered and in other cases it is asserted before the next floating point instruction is encountered depending upon the execution state of the instruction causing the exception.

The following class of floating point exceptions drive FERR# at the time the exception occurs (i.e., before encountering the next floating point instruction).

1. The stack fault, invalid operation, and denormal exceptions on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exceptions on store instructions (including integer store instructions).

The following class of floating point exceptions drive FERR# only after encountering the next floating point instruction.

1. Exceptions other than on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exception on all basic arithmetic, load, compare, and control instructions (i.e., all other instructions).

Ignore Numeric Error Input (IGNNE#)

The Intel486 DX microprocessor will ignore a numeric error and continue executing non-control floating point instructions when IGNNE# is asserted, but FERR# will still be activated. When deasserted, the Intel486 DX microprocessor will freeze on a non-control floating point instruction if a previous instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set.

The IGNNE# input is active LOW and is provided with a small internal pullup resistor. This input is asynchronous, but must meet setup and hold times t_{20} and t_{21} to insure recognition on any specific clock.

6.2.14 BUS SIZE CONTROL (BS16#, BS8#)

The BS16# and BS8# inputs allow external 16- and 8-bit busses to be supported with a small number of external components. The Intel486 DX CPU samples these pins every clock. The value sampled in the clock before ready determines the bus size. When asserting BS16# or BS8# only 16 or 8 bits of the data bus need be valid. If both BS16# and BS8# are asserted, an 8-bit bus width is selected.

When BS16# or BS8# are asserted the Intel486 DX microprocessor will convert a larger data request to the appropriate number of smaller transfers. The byte enables will also be modified appropriately for the bus size selected.

BS16# and BS8# are active LOW and are provided with small internal pullup resistors. BS16# and BS8# must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

6.2.15 ADDRESS BIT 20 MASK (A20M#)

Asserting the A20M# input causes the Intel486 DX microprocessor to mask physical address bit 20 before performing a lookup in the internal cache and before driving a memory cycle to the outside world. When A20M# is asserted, the Intel486 DX microprocessor emulates the 1 Mbyte address wrap-around that occurs on the 8086. A20M# is active LOW and must be asserted only when the processor is in real mode. The A20M# is not defined in Protected Mode. A20M# is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For correct operation of the chip, A20M# should be sampled high 2 clocks before and 2 clocks after RESET goes low.

6.2.16 BOUNDARY SCAN TEST SIGNALS

Test Clock (TCK)

TCK is an input to the Intel486 DX2 CPU and provides the clocking function required by the JTAG boundary scan feature. TCK is used to clock state information and data into and out of the component. State select information and data are clocked into the component on the rising edge of TCK on TMS and TDI, respectively. Data is clocked out of the part on the falling edge of TCK on TDO.

In addition to using TCK as a free running clock, it may be stopped in a low, O, state, indefinitely as described in IEEE 1149.1. While TCK is stopped in the low state, the boundary scan latches retain their state.

When boundary scan is not used, TCK should be tied high or left as a NC (This is important during power up to avoid the possibility of glitches on the TCK which could prematurely initiate boundary scan operations). TCK is supplied with an internal pullup resistor.

TCK is a clock signal and is used as a reference for sampling other JTAG signals. On the rising edge of TCK, TMS and TDI are sampled. On the falling edge of TCK, TDO is driven.

Test Mode Select (TMS)

TMS is decoded by the JTAG TAP (Tap Access Port) to select the operation of the test logic, as described in Section 8.5.4.

To guarantee deterministic behavior of the TAP controller, TMS is provided with an internal pull-up resistor. If boundary scan is not used, TMS may be tied high or left unconnected. TMS is sampled on the rising edge of TCK. TMS is used to select the internal TAP states required to load boundary scan instructions to data on TDI. For proper initialization of the JTAG logic, TMS should be driven high, "1", for at least four TCK cycles following the rising edge of RESET.

Test Data Input (TDI)

TDI is the serial input used to shift JTAG instructions and data into the component. The shifting of instructions and data occurs during the SHIFT-IR and SHIFT-DR controller states, respectively. These states are selected using the TMS signal as described in Section 8.5.4.

An internal pull-up resistor is provided on TDI to ensure a known logic state if an open circuit occurs on the TDI path. Note that when "1" is continuously shifted into the instruction register, the BYPASS instruction is selected. TDI is sampled on the rising edge of TCK, during the SHIFT-IR and the SHIFT-DR states. During all other TAP controller states, TDI is a "don't care".

Test Data Output (TDO)

TDO is the serial output used to shift JTAG instructions and data out of the component. The shifting of instructions and data occurs during the SHIFT-IR and SHIFT-DR TAP controller states, respectively. These states are selected using the TMS signal as described in Section 8.5.4. When not in SHIFT-IR or SHIFT-DR state, TDO is driven to a high impedance state to allow connecting TDO of different devices in parallel.

TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT-DR TAP controller states. At all other times TDO is driven to the high impedance state.

6.3 Write Buffers

The Intel486 DX2 microprocessor contains four write buffers to enhance the performance of consecutive writes to memory. The buffers can be filled at a rate of one write per clock until all four buffers are filled.

When all four buffers are empty and the bus is idle, a write request will propagate directly to the external bus bypassing the write buffers. If the bus is not available at the time the write is generated internally, the write will be placed in the write buffers and propagate to the bus as soon as the bus becomes available. The write is stored in the on-chip cache immediately if the write is a cache hit.

Writes will be driven onto the external bus in the same order in which they are received by the write buffers. Under certain conditions a memory read will go onto the external bus before the memory writes pending in the buffer even though the writes occurred earlier in the program execution.

A memory read will only be reordered in front of all writes in the buffers under the following conditions: If all writes pending in the buffers are cache hits and the read is a cache miss. Under these conditions the Intel486 DX microprocessor will not read from an external memory location that needs to be updated by one of the pending writes.

Reordering of a read with the writes pending in the buffers can only occur once before all the buffers are emptied. Reordering read once only maintains cache consistency. Consider the following example: The CPU writes to location X. Location X is in the internal cache, so it is updated there immediately. However, the bus is busy so the write out to main memory is buffered (see Figure 6.3(a)). At this point, any reads to location X would be cache hits and most up-to-date data would be read.

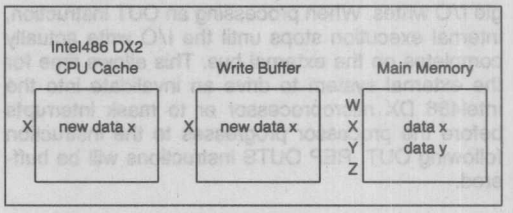


Figure 6.3(a)

The next instruction causes a read to location Y. Location Y is not in the cache (a cache miss). Since the write in the write buffer is a cache hit, the read is reordered. When location Y is read, it is put into the cache. The possibility exists that location Y will replace location X in the cache. If this is true, location X would no longer be cached (see Figure 6.3(b)).

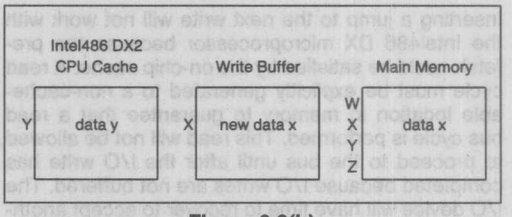


Figure 6.3(b)

Cache consistency has been maintained up to this point. If a subsequent read is to location X (now a cache miss) and it was reordered in front of the buffered write to location X, stale data would be read. This is why only 1 read is allowed to be reordered. Once a read is reordered, all the writes in the write buffer are flagged as cache misses to ensure that no more reads are reordered. Since one of the conditions to reorder a read is that all writes in the write buffer must be cache hits, no more reordering is allowed until all of those flagged writes propogate to the bus. Similarly, if an invalidation cycle is run all entries in the write buffer are flagged as cache misses.

For multiple processor systems and/or systems using DMA techniques, such as bus snooping, locked semaphores should be used to maintain cache consistency.

6.3.1 WRITE BUFFERS AND I/O CYCLES

Input/Output (I/O) cycles must be handled in a different manner by the write buffers.

I/O reads are never reordered in front of buffered memory writes. This insures that the Intel486 DX microprocessor will update all memory locations before reading status from an I/O device.

The Intel486 DX microprocessor never buffers single I/O writes. When processing an OUT instruction, internal execution stops until the I/O write actually completes on the external bus. This allows time for the external system to drive an invalidate into the Intel486 DX microprocessor or to mask interrupts before the processor progresses to the instruction following OUT. REP OUTS instructions will be buffered.

I/O device recovery time must be handled slightly differently by the Intel486 DX microprocessor than with the Intel386 microprocessor. I/O device back-to-back write recovery times could be guaranteed by the Intel386 microprocessor by inserting a jump to the next instruction in the code that writes to the device. The jump forces the Intel386 microprocessor to generate a prefetch bus cycle which can't begin until the I/O write completes.

Inserting a jump to the next write will not work with the Intel486 DX microprocessor because the prefetch could be satisfied by the on-chip cache. A read cycle must be explicitly generated to a non-cacheable location in memory to guarantee that a read bus cycle is performed. This read will not be allowed to proceed to the bus until after the I/O write has completed because I/O writes are not buffered. The I/O device will have time to recover to accept another write during the read cycle.

6.3.2 WRITE BUFFERS IMPLICATIONS ON LOCKED BUS CYCLES

Locked bus cycles are used for read-modify-write accesses to memory. During a read-modify-write access, a memory base variable is read, modified and then written back to the same memory location. It is important that no other bus cycles, generated by other bus masters or by the Intel486 DX microprocessor itself, be allowed on the external bus between the read and write portion of the locked sequence.

During a locked read cycle the Intel486 DX microprocessor will always access external memory, it will never look for the location in the on-chip cache, but for write cycles, data is written in the internal cache (if cache hit) and in the external memory. All data pending in the Intel486 DX microprocessor's write buffers will be written to memory before a locked cycle is allowed to proceed to the external bus.

The Intel486 DX microprocessor will assert the LOCK# pin after the write buffers are emptied during a locked bus cycle. With the LOCK# pin asserted, the microprocessor will read the data, operate on the data and place the results in a write buffer. The contents of the write buffer will then be written to external memory. LOCK# will become inactive after the write part of the locked cycle.

6.4 Interrupt and Non-Maskable Interrupt Interface

The Intel486 DX microprocessor provides two asynchronous interrupt inputs, INTR (interrupt request) and NMI (non-maskable interrupt input). This section describes the hardware interface between the instruction execution unit and the pins. For a description of the algorithmic response to interrupts refer to Section 2.7. For interrupt timings refer to Section 7.2.10.

6.4.1 INTERRUPT LOGIC

The Intel486 DX microprocessor contains a two-clock synchronizer on the interrupt line. An interrupt request will reach the internal instruction execution unit two clocks after the INTR pin is asserted, if proper setup is provided to the first stage of the synchronizer.

There is no special logic in the interrupt path other than the synchronizer. The INTR signal is level sensitive and must remain active for the instruction execution unit to recognize it. The interrupt will not be serviced by the Intel486 DX microprocessor if the INTR signal does not remain active.

The instruction execution unit will look at the state of the synchronized interrupt signal at specific clocks during the execution of instructions (if interrupts are enabled). These specific clocks are at instruction boundaries, or iteration boundaries in the case of string move instructions. Interrupts will only be accepted at these boundaries.

An interrupt must be presented to the Intel486 DX microprocessor INTR pin three clocks before the end of an instruction for the interrupt to be acknowledged. Presenting the interrupt 3 clocks before the end of an instruction allows the interrupt to pass through the two clock synchronizer leaving one clock to prevent the initiation of the next sequential instruction and to begin interrupt service. If the interrupt is not received in time to prevent the next instruction, it will be accepted at the end of next instruction, assuming INTR is still held active. The interrupt service microcode will start after two dead clocks.

The longest latency between when an interrupt request is presented on the INTR pin and when the interrupt service begins is: longest instruction used + the two clocks for synchronization + one clock required to vector into the interrupt service microcode.

6.4.2 NMI LOGIC

The NMI pin has a synchronizer like that used on the INTR line. Other than the synchronizer, the NMI logic is different from that of the maskable interrupt.

NMI is edge triggered as opposed to the level triggered INTR signal. The rising edge of the NMI signal is used to generate the interrupt request. The NMI input need not remain active until the interrupt is actually serviced. The NMI pin only needs to remain active for a single clock if the required setup and hold times are met. NMI will operate properly if it is held active for an arbitrary number of clocks.

The NMI input must be held inactive for at least four clocks after it is asserted to reset the edge triggered logic. A subsequent NMI may not be generated if the NMI is not held inactive for at least two clocks after being asserted.

The NMI input is internally masked whenever the NMI routine is entered. The NMI input will remain masked until an IRET (return from interrupt) instruction is executed. Masking the NMI signal prevents recursive NMI calls. If another NMI occurs while the NMI is masked off, the pending NMI will be executed after the current NMI is done. Only one NMI can be pending while NMI is masked.

6.5 Reset and Initialization

The Intel486 DX2 microprocessor has a built in self test (BIST) that can be run during reset. The BIST is invoked if the AHOLD pin is asserted for 2 clocks before and 2 clocks after RESET is deasserted. RESET must be active for 15 clocks with or without BIST enabled. Refer to Section 8.0 for information on Intel486 DX2 microprocessor testability. To ensure proper results, FLUSH# must not be asserted while BIST is executing.

The Intel486 DX microprocessor registers have the values shown in Table 1.5 after RESET is performed. The EAX register contains information on the success or failure of the BIST if the self test is executed. The DX register always contains a component identifier at the conclusion of RESET. The upper byte of DX (DH) will contain 04 and the lower byte (DL) will contain a stepping identifier (see Table 1.5). The floating point registers are initialized as if the FINIT/FNINIT (initialize processor) instruction was executed if the BIST was performed. If the BIST is not executed, the floating point registers are unchanged.

Table 6.2. Register Values after Reset

Register	Initial Value (BIST)	Initial Value (No Bist)
EAX	Zero (Pass)	Undefined
ECX	Undefined	Undefined
EDX	0400+Revision ID	0400+Revision ID
EBX	Undefined	Undefined
ESP	Undefined	Undefined
EBP	Undefined	Undefined
ESI	Undefined	Undefined
EDI	Undefined	Undefined
EFLAGS	00000002h	00000002h
EIP	0FFF0h	0FFF0h
ES	0000h	0000h
CS	F000h*	F000h*
SS	0000h	0000h
DS	0000h	0000h
FS	0000h	0000h
GS	0000h	0000h
IDTR	Base=0, Limit=3FFh	Base=0, Limit=3FFh
CR0	60000010h	60000010h
DR7	00000000h	00000000h
CW	037Fh	Unchanged
SW	0000h	Unchanged
TW	FFFFh	Unchanged
FIP	00000000h	Unchanged
FEA	00000000h	Unchanged
FCS	0000h	Unchanged
FDS	0000h	Unchanged
FOP	000h	Unchanged
FSTACK	Undefined	Unchanged

Table 6.3. Component and Revision ID

Intel486 DX2 CPU Stepping Name	Component ID	Revision ID
A	04h	32h
B	04h	33h

The Intel486 DX microprocessor will start executing instructions at location FFFFFFF0H after RESET. When the first InterSegment Jump or Call is executed, address lines A20–A31 will drop LOW for CS-relative memory cycles, and the Intel486 DX microprocessor will only execute instructions in the lower one Mbyte of physical memory. This allows the system designer to use a ROM at the top of physical memory to initialize the system and take care of RESETs.

RESET forces the Intel486 DX microprocessor to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active.

All entries in the cache are invalidated by RESET.

6.5.1 PIN STATE DURING RESET

The Intel486 DX2 microprocessor recognizes and can respond to HOLD, AHOLD, and BOFF# requests regardless of the state of RESET. Thus, even though the processor is in reset, it can still float its bus in response to any of these requests.

While in reset, the Intel486 DX microprocessor bus is in the state shown in Figure 6.4 if the HOLD, AHOLD and BOFF# requests are inactive. Note that the address (A31–A2, BE3#–BE0#) and cycle definition (M/IO#, D/C#, W/R#) pins are undefined from the time reset is asserted up to the start of the first bus cycle. All undefined pins (except FERR#) assume known values at the beginning of the first bus cycle. The first bus cycle is always a code fetch to address FFFFFFF0H. FERR# reflects the state of the ES (error summary status) bit in the floating point unit status word. The ES bit is initialized whenever the floating point unit state is initialized. The floating point unit's status word register can be initialized by BIST or by executing FINIT/FNINIT instruction. Thus, after reset and before executing the first FINIT or FNINIT instruction, the values of the FERR# and the numeric status word register bits 0–7 depends on whether or not BIST is performed. Table 6-4 shows the state of FERR# signal after reset and before the execution of the FINIT/FNINIT instruction.

Table 6.4

BIST Performed	FERR# Pin	FPU Status Word Register Bits 0–7
YES	Inactive (High)	Inactive (Low)
NO	Undefined (Low or High)	Undefined (Low or High)

After the first FINIT or FNINIT instruction, FERR# pin and the FPU status word register bits (0–7) will be inactive irrespective of the Built-In Self-Test (BIST).

Power Down Mode (Upgrade Processor Support)

The Power Down Mode on the Intel486 DX2 microprocessor, when initiated by the upgrade processor, reduces the power consumption of the Intel486 DX2 CPU (see Table 14-2 D.C. Specifications), as well as forces all of its output signals to be 3-stated. The UP# pin on the Intel486 DX2 microprocessor is used for enabling the Power Down Mode.

Once the UP# pin is driven active by the upgrade processor upon power-up, the Intel486 DX2 microprocessor's bus is floated immediately. The Intel486 DX2 CPU enters the Power Down Mode when the UP# pin is sampled asserted in the clock before the falling edge of RESET. The UP# pin has no effect on the power down status, except during this edge. The Intel486 DX2 CPU then remains in the Power Down Mode until the next time the RESET signal is activated. For warm resets, with the upgrade processor in the system, the Intel486 DX2 CPU will remain 3-stated and re-enter the Power Down Mode once RESET is de-asserted. Similarly for power-up resets, if the upgrade processor is not taken out of the system, the Intel486 DX2 CPU will 3-state its outputs upon sensing the UP# pin active and enter the Power Down Mode after the falling edge of RESET.

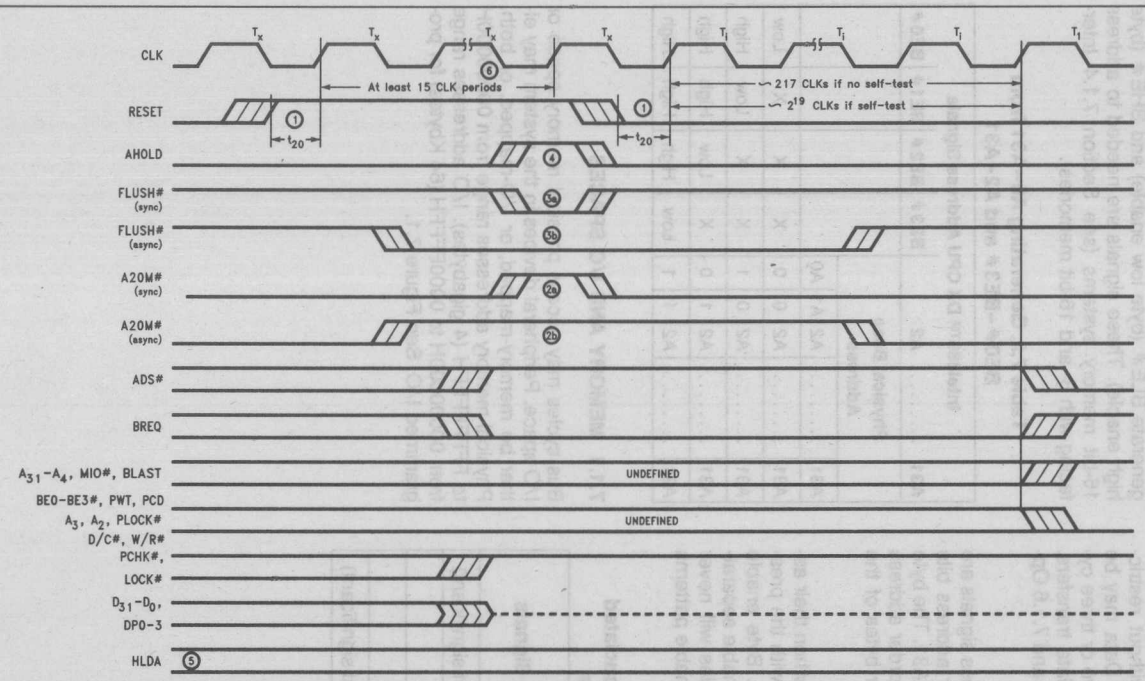


Figure 6.4. Pin States during RESET

NOTES:

1. RESET is an asynchronous input. t_{20} must be met only to guarantee recognition on a specific clock edge.
2a. When A20M# is driven synchronously, it must be driven high (inactive) for the CLK edge prior to the falling edge of RESET to ensure proper operation. A20M# setup and hold times must be met.
2b. When A20M# is driven asynchronously, it must be driven high (inactive) for two CLKs prior to and two CLKs after the falling edge of RESET to ensure proper operation.
3a. When FLUSH# is driven synchronously, it should be driven low (active) for the CLK edge prior to the falling edge of RESET to invoke the 3-state Output Test Mode. All outputs are guaranteed 3-stated within 10 CLKs of RESET being deasserted. FLUSH# setup and hold times must be met.
3b. When FLUSH# is driven asynchronously, it must be driven low (active) for two CLKs prior to and two CLKs after the falling edge of RESET to invoke the 3-state Output Test Mode. All outputs are guaranteed 3-stated within 10 CLKs of RESET being deasserted.
4. AHOLD should be driven high (active) for the CLK edge prior to the falling edge of RESET to invoke the Built-In-Self-Test (BIST). AHOLD setup and hold times must be met.
5. Hold is recognized normally during RESET. On power-up HLDA is indeterminate until RESET is recognized by the CPU.
6. 15 CLKs RESET pulse width for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.

7.0 BUS OPERATION

7.1 Data Transfer Mechanism

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte, word and dword lengths may be transferred without restrictions on physical address alignment. Data may be accessed at any byte boundary but two or three cycles may be required for unaligned data transfers. See Section 7.1.3 Dynamic Bus Sizing and 7.1.6 Operand Alignment.

The Intel486 DX microprocessor address signals are split into two components. High-order address bits are provided by the address lines, A2–A31. The byte enables, BE0#–BE3#, form the low-order address and provide linear selects for the four bytes of the 32-bit address bus.

The byte enable outputs are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 7.1. Byte enable patterns which have a negated byte enable separating two or three asserted byte enables will never occur (see Table 7.5). All other byte enable patterns are possible.

Table 7.1. Byte Enables and Associated Data and Operand Bytes

Byte Enable Signal	Associated Data Bus Signals	
BE0#	D0–D7	(byte 0—least significant)
BE1#	D8–D15	(byte 1)
BE2#	D16–D23	(byte 2)
BE3#	D24–D31	(byte 3—most significant)

Address bits A0 and A1 of the physical operand's base address can be created when necessary. Use of the byte enables to create A0 and A1 is shown in Table 7.2. The byte enables can also be decoded to generate BLE# (byte low enable) and BHE# (byte high enable). These signals are needed to address 16-bit memory systems (see Section 7.1.4 Interfacing with 8- and 16-bit memories).

Table 7.2. Generating A0–A31 from BE0#–BE3# and A2–A31

Intel486™ DX CPU Address Signals								
A31	.....	A2			BE3#	BE2#	BE1#	BE0#
Physical Base Address								
A31	.....	A2	A1	A0				
A31	.....	A2	0	0	X	X	X	Low
A31	.....	A2	0	1	X	X	Low	High
A31	.....	A2	1	0	X	Low	High	High
A31	.....	A2	1	1	Low	High	High	High

7.1.1 MEMORY AND I/O SPACES

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. Physical memory addresses range from 00000000H to FFFFFFFFH (4 gigabytes). I/O addresses range from 00000000H to 0000FFFFH (64 Kbytes) for programmed I/O. See Figure 7.1.

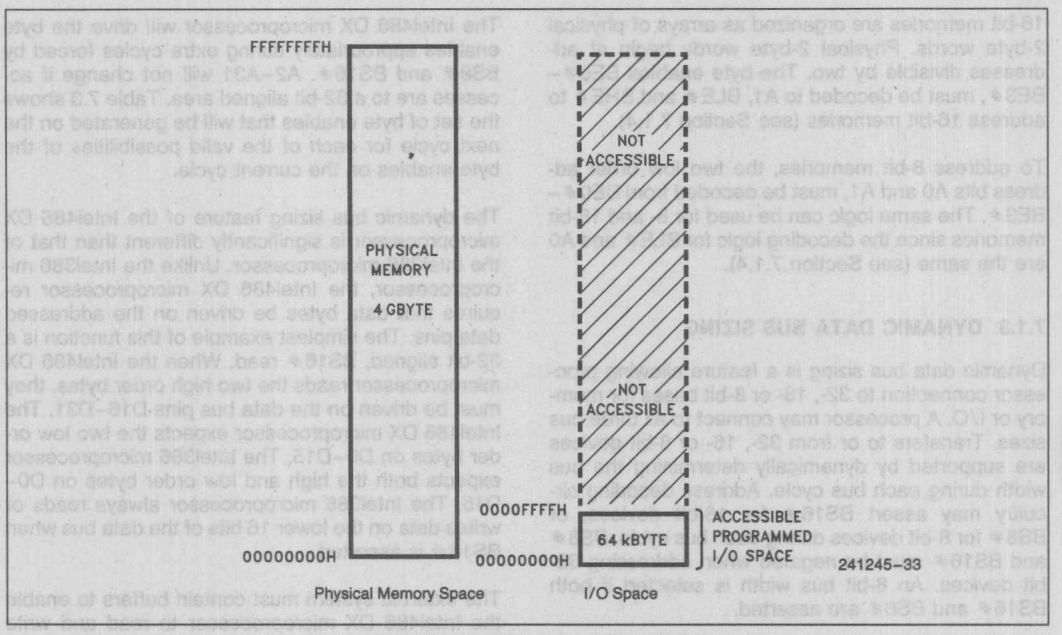


Figure 7.1. Physical Memory and I/O Spaces

7.1.2 MEMORY AND I/O SPACE ORGANIZATION

The Intel486 DX microprocessor data path to memory and input/output (I/O) spaces can be 32-, 16- or 8-bits wide. The byte enable signals, BE0#–BE3#, allow byte granularity when addressing any memory or I/O structure whether 8, 16 or 32 bits wide.

The Intel486 DX microprocessor includes bus control pins, BS16# and BS8#, which allow direct connection to 16- and 8-bit memories and I/O devices. Cycles to 32-, 16- and 8-bit may occur in any sequence, since the BS8# and BS16# signals are sampled during each bus cycle.

32-bit wide memory and I/O spaces are organized as arrays of physical 4-byte words. Each memory or I/O 4-byte word has four individually addressable bytes at consecutive byte addresses (see Figure 7.2). The lowest addressed byte is associated with data signals D0–D7; the highest-addressed byte with D24–D31. Physical 4-byte words begin at addresses divisible by four.

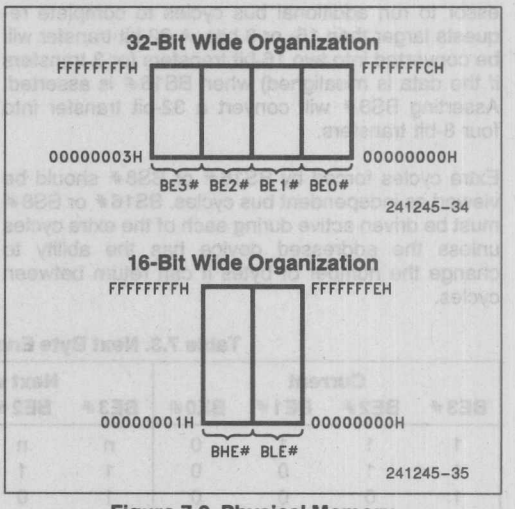


Figure 7.2. Physical Memory and I/O Space Organization

16-bit memories are organized as arrays of physical 2-byte words. Physical 2-byte words begin at addresses divisible by two. The byte enables BE0# – BE3#, must be decoded to A1, BLE# and BHE# to address 16-bit memories (see Section 7.1.4).

To address 8-bit memories, the two low order address bits A0 and A1, must be decoded from BE0# – BE3#. The same logic can be used for 8- and 16-bit memories since the decoding logic for BLE# and A0 are the same (see Section 7.1.4).

7.1.3 DYNAMIC DATA BUS SIZING

Dynamic data bus sizing is a feature allowing processor connection to 32-, 16- or 8-bit buses for memory or I/O. A processor may connect to all three bus sizes. Transfers to or from 32-, 16- or 8-bit devices are supported by dynamically determining the bus width during each bus cycle. Address decoding circuitry may assert BS16# for 16-bit devices, or BS8# for 8-bit devices during each bus cycle. BS8# and BS16# must be negated when addressing 32-bit devices. An 8-bit bus width is selected if both BS16# and BS8# are asserted.

BS16# and BS8# force the Intel486 DX microprocessor to run additional bus cycles to complete requests larger than 16- or 8 bits. A 32-bit transfer will be converted into two 16-bit transfers (or 3 transfers if the data is misaligned) when BS16# is asserted. Asserting BS8# will convert a 32-bit transfer into four 8-bit transfers.

Extra cycles forced by BS16# or BS8# should be viewed as independent bus cycles. BS16# or BS8# must be driven active during each of the extra cycles unless the addressed device has the ability to change the number of bytes it can return between cycles.

The Intel486 DX microprocessor will drive the byte enables appropriately during extra cycles forced by BS8# and BS16#. A2–A31 will not change if accesses are to a 32-bit aligned area. Table 7.3 shows the set of byte enables that will be generated on the next cycle for each of the valid possibilities of the byte enables on the current cycle.

The dynamic bus sizing feature of the Intel486 DX microprocessor is significantly different than that of the Intel386 microprocessor. Unlike the Intel386 microprocessor, the Intel486 DX microprocessor requires that data bytes be driven on the addressed data pins. The simplest example of this function is a 32-bit aligned, BS16# read. When the Intel486 DX microprocessor reads the two high order bytes, they must be driven on the data bus pins D16–D31. The Intel486 DX microprocessor expects the two low order bytes on D0–D15. The Intel386 microprocessor expects both the high and low order bytes on D0–D15. The Intel386 microprocessor always reads or writes data on the lower 16 bits of the data bus when BS16# is asserted.

The external system must contain buffers to enable the Intel486 DX microprocessor to read and write data on the appropriate data bus pins. Table 7.4 shows the data bus lines where the Intel486 DX microprocessor expects data to be returned for each valid combination of byte enables and bus sizing options.

Valid data will only be driven onto data bus pins corresponding to active byte enables during write cycles. Other pins in the data bus will be driven but they will not contain valid data. Unlike the Intel386 microprocessor, the Intel486 DX microprocessor will not duplicate write data onto parts of the data bus for which the corresponding byte enable is negated.

Table 7.3. Next Byte Enable Values for BSn# Cycles

Current				Next with BS8#				Next with BS16#			
BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#
1	1	1	0	n	n	n	n	n	n	n	n
1	1	0	0	1	1	0	1	n	n	n	n
1	0	0	0	1	0	0	1	1	0	1	1
0	0	0	0	0	0	0	1	0	0	1	1
1	1	0	1	n	n	n	n	n	n	n	n
1	0	0	1	1	0	1	1	1	0	1	1
0	0	0	1	0	0	1	1	0	0	1	1
1	0	1	1	n	n	n	n	n	n	n	n
0	0	1	1	0	1	1	1	n	n	n	n
0	1	1	1	n	n	n	n	n	n	n	n

"n" means that another bus cycle will not be required to satisfy the request.

Table 7.4. Data Pins Read with Different Bus Sizes

BE3#	BE2#	BE1#	BE0#	w/o BS8#/BS16#	w BS8#	W BS16#
1	1	1	0	D7–D0	D7–D0	D7–D0
1	1	0	0	D15–D0	D7–D0	D15–D0
1	0	0	0	D23–D0	D7–D0	D15–D0
0	0	0	0	D31–D0	D7–D0	D15–D0
1	1	0	1	D15–D8	D15–D8	D15–D8
1	0	0	1	D23–D8	D15–D8	D15–D8
0	0	0	1	D31–D8	D15–D8	D15–D8
1	0	1	1	D23–D16	D23–D16	D23–D16
0	0	1	1	D31–D16	D23–D16	D31–D16
0	1	1	1	D31–D24	D31–D24	D31–D24

7.1.4 INTERFACING WITH 8-, 16- AND 32-BIT MEMORIES

In 32-bit physical memories such as Figure 7.3, each 4-byte word begins at a byte address that is a multiple of four. A2–A31 are used as a 4-byte word select. BE0#–BE3# select individual bytes within the 4-byte word. BS8# and BS16# are negated for all bus cycles involving the 32-bit array.

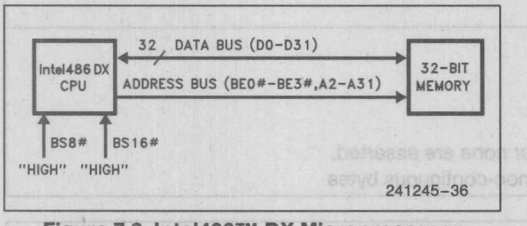


Figure 7.3. Intel486™ DX Microprocessor with 32-Bit Memory

16- and 8-bit memories require external byte swapping logic for routing data to the appropriate data lines and logic for generating BHE#, BLE# and A1. In systems where mixed memory widths are used, extra address decoding logic is necessary to assert BS16# or BS8#.

Figure 7.4 shows the Intel486 DX microprocessor address bus interface to 32-, 16- and 8-bit memories. To address 16-bit memories the byte enables must be decoded to produce A1, BHE# and BLE# (A0). For 8-bit wide memories the byte enables must be decoded to produce A0 and A1. The same byte select logic can be used in 16- and 8-bit systems since BLE# is exactly the same as A0 (see Table 7.5).

BE0#–BE3# can be decoded as shown in Table 7.5 to generate A1, BHE# and BLE#. The byte select logic necessary to generate BHE# and BLE# is shown in Figure 7.5.

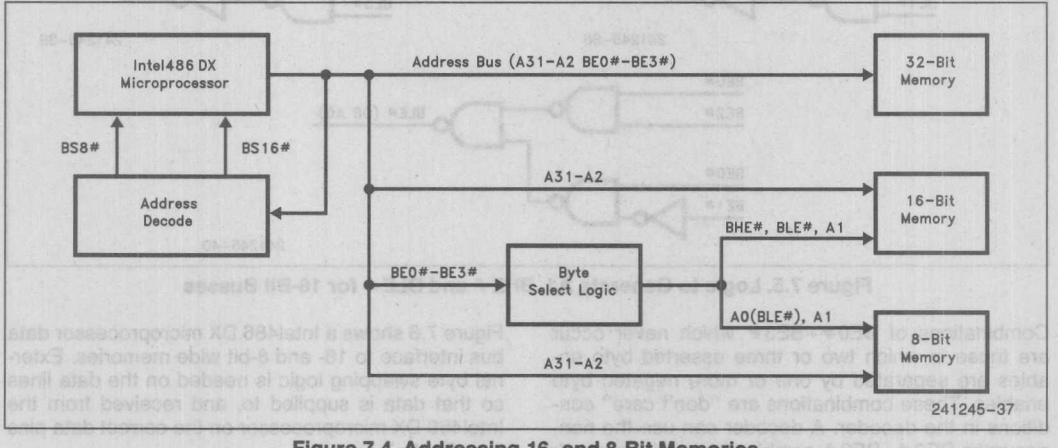


Figure 7.4. Addressing 16- and 8-Bit Memories

Table 7.5. Generating A1, BHE# and BLE# for Addressing 16-Bit Devices

Intel486™ DX CPU Signals				8, 16-Bit Bus Signals			Comments
BE3#	BE2#	BE1#	BE0#	A1	BHE#	BLE# (A0)	
H*	H*	H*	H*	x	x	x	x—no active bytes
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	
H*	L*	H*	L*	x	x	x	x—not contiguous bytes
H	L	L	H	L	L	H	
H	L	L	L	L	L	L	
L	H	H	H	H	L	H	
L*	H*	H*	L*	x	x	x	x—not contiguous bytes
L*	H*	L*	H*	x	x	x	x—not contiguous bytes
L*	H*	L*	L*	x	x	x	x—not contiguous bytes
L	L	H	H	H	L	L	
L*	L*	H*	L*	x	x	x	x—not contiguous bytes
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

BLE# asserted when D0–D7 of 16-bit bus is active.
BHE# asserted when D8–D15 of 16-bit bus is active.
A1 low for all even words; A1 high for all odd words.

Key:
x = don't care
H = high voltage level
L = low voltage level
* = a non-occurring pattern of Byte Enables; either none are asserted, or the pattern has Byte Enables asserted for non-contiguous bytes

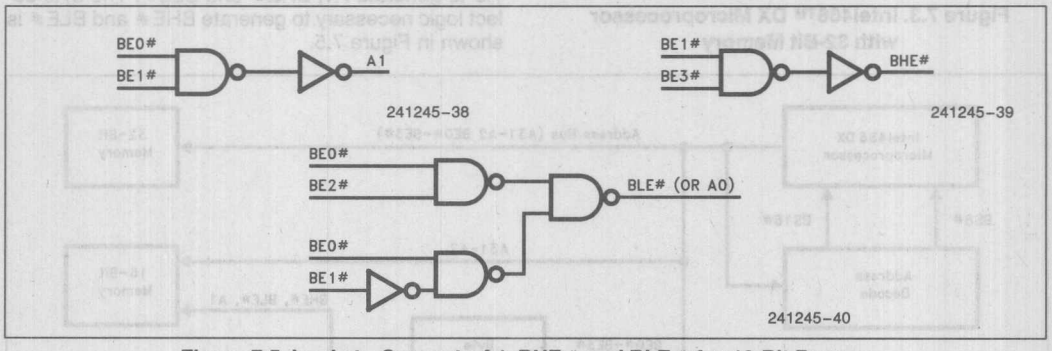


Figure 7.5. Logic to Generate A1, BHE# and BLE# for 16-Bit Busses

Combinations of BE0#–BE3# which never occur are those in which two or three asserted byte enables are separated by one or more negated byte enables. These combinations are "don't care" conditions in the decoder. A decoder can use the non-occurring BE0#–BE3# combinations to its best advantage.

Figure 7.6 shows a Intel486 DX microprocessor data bus interface to 16- and 8-bit wide memories. External byte swapping logic is needed on the data lines so that data is supplied to, and received from the Intel486 DX microprocessor on the correct data pins (see Table 7.4).

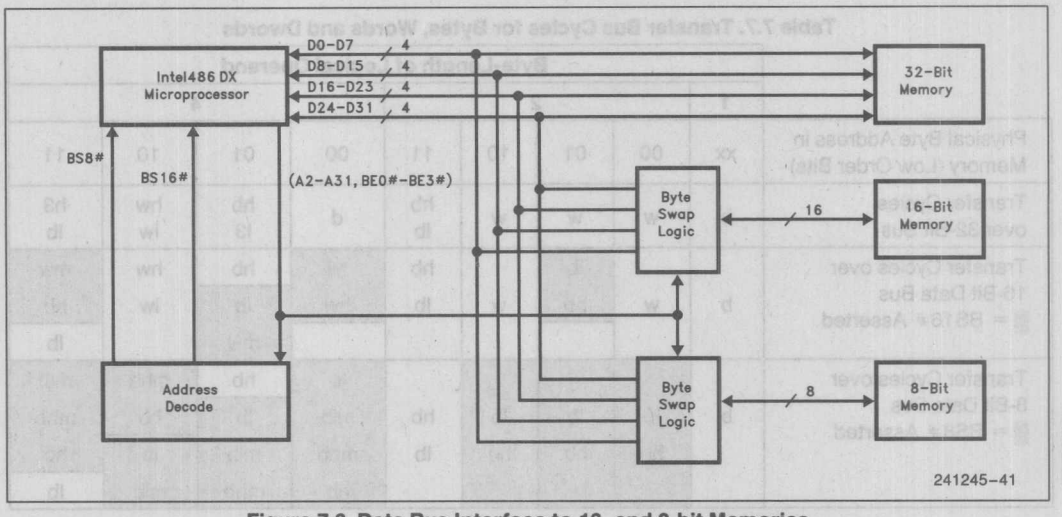


Figure 7.6. Data Bus Interface to 16- and 8-bit Memories

7.1.5 DYNAMIC BUS SIZING DURING CACHE LINE FILLS

BS8# and BS16# can be driven during cache line fills. The Intel486 DX microprocessor will generate enough 8- or 16-bit cycles to fill the cache line. This can be up to 16 8-bit cycles.

The external system should assume that all byte enables are active for the first cycle of a cache line fill. The Intel486 DX microprocessor will generate proper byte enables for subsequent cycles in the line fill. Table 7.6 shows the appropriate A0 (BLE#), A1 and BHE# for the various combinations of the Intel486 DX microprocessor byte enables on both the first and subsequent cycles of the cache line fill. The "*" marks all combinations of byte enables that will be generated by the Intel486 DX microprocessor during a cache line fill.

7.1.6 OPERAND ALIGNMENT

Physical 4-byte words begin at addresses that are multiples of four. It is possible to transfer a logical operand that spans more than one physical 4-byte word of memory or I/O at the expense of extra cycles. Examples are 4-byte operands beginning at addresses that are not evenly divisible by 4, or 2-byte words split between two physical 4-byte words. These are referred to as unaligned transfers.

Operand alignment and data bus size dictate when multiple bus cycles are required. Table 7.7 describes the transfer cycles generated for all combinations of logical operand lengths, alignment, and data bus sizing. When multiple cycles are required to transfer a multi-byte logical operand, the highest-order bytes are transferred first. For example, when the processor does a 4-byte unaligned read beginning at location x11 in the 4-byte aligned space, the three high order bytes are read in the first bus cycle. The low byte is read in a subsequent bus cycle.

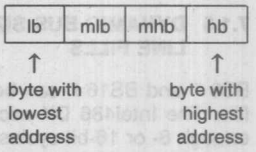
Table 7.6. Generating A0, A1 and BHE# from the Intel486™ DX Microprocessor Byte Enables

BE3#	BE2#	BE1#	BE0#	First Cache Fill Cycle			Any Other Cycle		
				A0	A1	BHE#	A0	A1	BHE#
1	1	1	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
*0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	1	0	0
1	0	0	1	0	0	0	1	0	0
*0	0	0	1	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	1
*0	0	1	1	0	0	0	0	1	0
*0	1	1	1	0	0	0	1	1	0

Table 7.7. Transfer Bus Cycles for Bytes, Words and Dwords

	Byte-Length of Logical Operand								
	1	2				4			
Physical Byte Address in Memory (Low Order Bits)	xx	00	01	10	11	00	01	10	11
Transfer Cycles over 32-Bit Bus	b	w	w	w	hb lb	d	hb l3	hw lw	h3 lb
Transfer Cycles over 16-Bit Data Bus = BS16# Asserted	b	w	lb hb	w	hb lb	lw hw	hb lb mw	hw lw	mw hb lb
Transfer Cycles over 8-Bit Data Bus = BS8# Asserted	b	lb hb	lb hb	lb hb	hb lb	lb mlb mhb hb	hb lb mlb mhb	mhb hb lb mlb	mlb mhb hb lb

KEY:
b = byte transfer
w = 2-byte transfer
3 = 3-byte transfer
d = 4-byte transfer
h = high-order portion
l = low-order portion
m = mid-order portion



The function of unaligned transfers with dynamic bus sizing is not obvious. When the external systems asserts BS16# or BS8# forcing extra cycles, low-order bytes or words are transferred first (opposite to the example above). When the Intel486 DX microprocessor requests a 4-byte read and the external system asserts BS16#, the lower 2 bytes are read first followed by the upper 2 bytes.

In the unaligned transfer described above, the processor requested three bytes on the first cycle. If the external system asserted BS16# during this 3-byte transfer, the lower word is transferred first followed by the upper byte. In the final cycle the lower byte of the 4-byte operand is transferred as in the 32-bit example above.

7.2 Bus Functional Description

The Intel486 DX microprocessor supports a wide variety of bus transfers to meet the needs of high performance systems. Bus transfers can be single cycle or multiple cycle, burst or non-burst, cacheable or non-cacheable, 8-, 16- or 32-bit, and pseudo-locked. To support multiprocessing systems there are cache invalidation cycles and locked cycles.

This section begins with basic non-cacheable non-burst single cycle transfers. It moves on to multiple cycle transfers and introduces the burst mode. Cacheability is introduced in Section 7.2.3. The remaining sections describe locked, pseudo-locked, invalidate, bus hold and interrupt cycles.

Bus cycles and data cycles are discussed in this section. A bus cycle is at least two clocks long and begins with ADS# active in the first clock and ready active in the last clock. Data is transferred to or from the Intel486 DX microprocessor during a data cycle. A bus cycle contains one or more data cycles.

Refer to Section 7.2.13 for a description of the bus states shown in the timing diagrams.

7.2.1 NON-CACHEABLE NON-BURST SINGLE CYCLE

7.2.1.1 No Wait States

The fastest non-burst bus cycle that the Intel486 DX microprocessor supports is two clocks long. These cycles are called 2-2 cycles because reads and writes take two cycles each. The first 2 refers to

reads and the second to writes. For example, if a wait state needs to be added to a write, the cycle would be called 2-3.

Basic two clock read and write cycles are shown in Figure 7.7. The Intel486 DX microprocessor initiates a cycle by asserting the address status signal (ADS#) at the rising edge of the first clock. The ADS# output indicates that a valid bus cycle definition and address is available on the cycle definition lines and address bus.

The non-burst ready input (RDY#) is returned by the external system in the second clock. RDY# indicates that the external system has presented valid data on the data pins in response to a read or the external system has accepted data in response to a write.

The Intel486 DX microprocessor samples RDY# at the end of the second clock. The cycle is complete if RDY# is active (LOW) when sampled. Note that RDY# is ignored at the end of the first clock of the bus cycle.

The burst last signal (BLAST#) is asserted (LOW) by the Intel486 DX microprocessor during the second clock of the first cycle in all bus transfers illustrated in Figure 7.7. This indicates that each transfer is complete after a single cycle. The Intel486 DX microprocessor asserts BLAST# in the last cycle of a bus transfer.

The timing of the parity check output (PCHK#) is shown in Figure 7.7. The Intel486 DX microprocessor drives the PCHK# output one clock after ready terminates a read cycle. PCHK# indicates the parity status for the data sampled at the end of the previous clock. The PCHK# signal can be used by the external system. The Intel486 DX microprocessor does nothing in response to the PCHK# output.

7.2.1.2 Inserting Wait States

The external system can insert wait states into the basic 2-2 cycle by driving RDY# inactive at the end of the second clock. RDY# must be driven inactive to insert a wait state. Figure 7.8 illustrates a simple non-burst, non-cacheable signal with one wait state added. Any number of wait states can be added to an Intel486 DX microprocessor bus cycle by maintaining RDY# inactive.

The burst ready input (BRDY#) must be driven inactive on all clock edges where RDY# is driven inactive for proper operation of these simple non-burst cycles.

7.2.2 MULTIPLE AND BURST CYCLE BUS TRANSFERS

Multiple cycle bus transfers can be caused by internal requests from the Intel486 DX microprocessor or by the external memory system. An internal request for a 64-bit floating point load or a 128-bit pre-fetch must take more than one cycle. Internal requests for unaligned data may also require multiple bus cycles. A cache line fill requires multiple cycles to complete. The external system can cause a multiple cycle transfer when it can only supply 8 or 16 bits per cycle.

Only multiple cycle transfers caused by internal requests are considered in this section. Cacheable cycles and 8- and 16-bit transfers are covered in Sections 7.2.3 and 7.2.5.

7.2.2.1 Burst Cycles

The Intel486 DX microprocessor can accept burst cycles for any bus requests that require more than a single data cycle. During burst cycles, a new data item is strobed into the Intel486 DX microprocessor every clock rather than every other clock as in non-burst cycles. The fastest burst cycle requires 2 clocks for the first data item with subsequent data items returned every clock.

The Intel486 DX microprocessor is capable of bursting a maximum of 32 bits during a write. Burst writes can only occur if BS8# or BS16# is asserted. For example, the Intel486 DX microprocessor can burst write four 8-bit operands or two 16-bit operands in a single burst cycle. But the Intel486 DX microprocessor cannot burst multiple 32-bit writes in a single burst cycle.

Burst cycles begin with the Intel486 DX microprocessor driving out an address and asserting ADS# in the same manner as non-burst cycles. The Intel486 DX microprocessor indicates that it is willing to perform a burst cycle by holding the burst last signal (BLAST#) inactive in the second clock of the cycle. The external system indicates its willingness to do a burst cycle by returning the burst ready signal (BRDY#) active.

The addresses of the data items in a burst cycle will all fall within the same 16-byte aligned area (corresponding to an internal Intel486 DX microprocessor cache line). A 16-byte aligned area begins at location XXXXXXX0 and ends at location XXXXXXXF. During a burst cycle, only BE0–3#, A_2, and A_3 may change. A_4–A_{31}, M/IO#, D/C#, and W/R# will remain stable throughout a burst. Given the first address in a burst, external hardware can easily calculate the address of subsequent transfers in advance. An external memory system can be designed to quickly fill the Intel486 DX microprocessor internal cache lines.

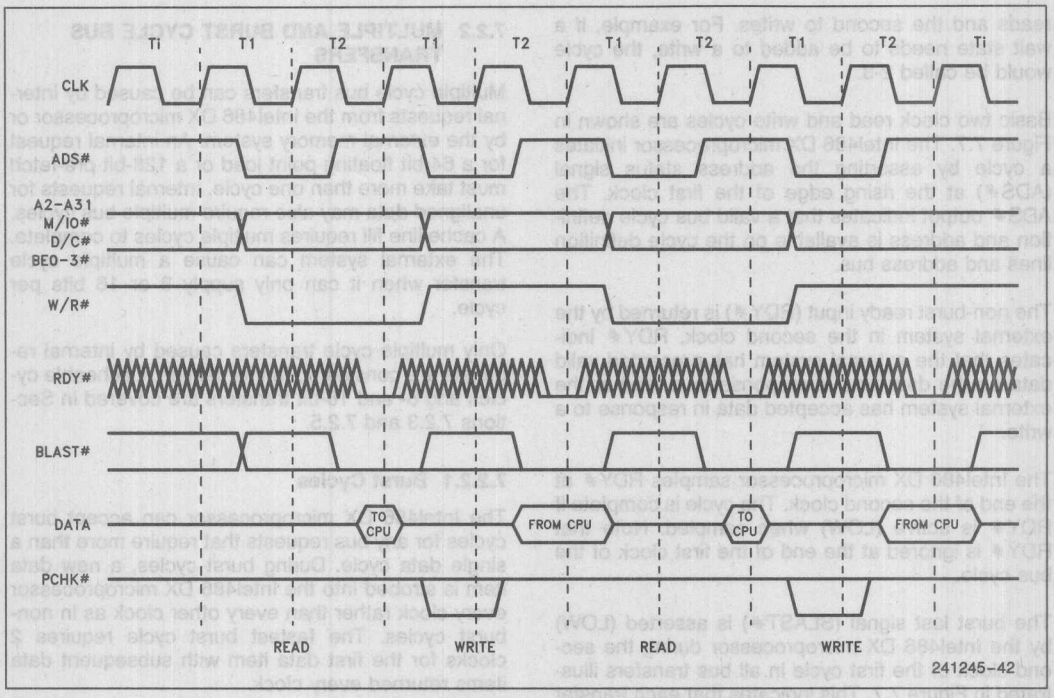


Figure 7.7. Basic 2-2 Bus Cycle

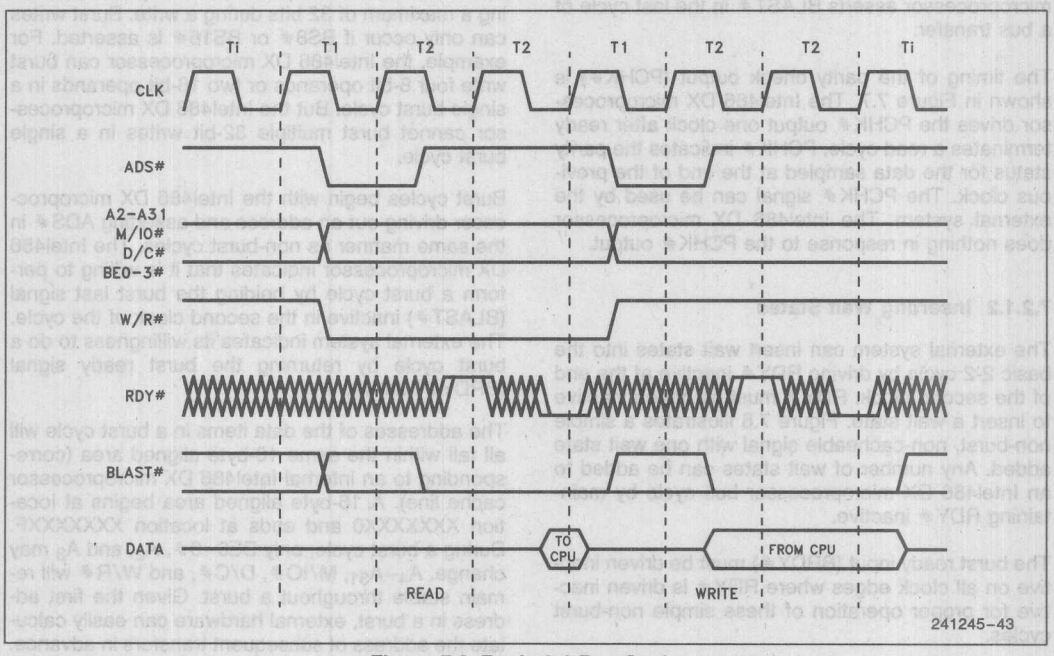


Figure 7.8. Basic 3-3 Bus Cycle

Burst cycles are not limited to cache line fills. Any multiple cycle read request by the Intel486 DX microprocessor can be converted into a burst cycle. The Intel486 DX microprocessor will only burst the number of bytes needed to complete a transfer. For example, eight bytes will be bursted in for a 64-bit floating point non-cacheable read.

The external system converts a multiple cycle request into a burst cycle by returning BRDY# active rather than RDY# (non-burst ready) in the first cycle of a transfer. For cycles that cannot be bursted such as interrupt acknowledge and halt, BRDY# has the same effect as RDY#. BRDY# is ignored if both BRDY# and RDY# are returned in the same clock. Memory areas and peripheral devices that cannot perform bursting must terminate cycles with RDY#.

7.2.2.2 Terminating Multiple and Burst Cycle Transfers

The Intel486 DX microprocessor drives BLAST# inactive for all but the last cycle in a multiple cycle transfer. BLAST# is driven inactive in the first cycle to inform the external system that the transfer could take additional cycles. BLAST# is driven active in the last cycle of the transfer indicating that the next time BRDY# or RDY# is returned the transfer is complete.

BLAST# is not valid in the first clock of a bus cycle. It should be sampled only in the second and subsequent clocks when RDY# or BRDY# is returned.

The number of cycles in a transfer is a function of several factors including the number of bytes the microprocessor needs to complete an internal request (1, 2, 4, 8, or 16), the state of the bus size inputs (BS8# and BS16#), the state of the cache enable input (KEN#) and alignment of the data to be transferred.

When the Intel486 DX microprocessor initiates a request it knows how many bytes will be transferred and if the data is aligned. The external system must tell the microprocessor whether the data is cacheable (if the transfer is a read) and the width of the bus by returning the state of the KEN#, BS8# and BS16# inputs one clock before RDY# or BRDY# is returned. The Intel486 DX microprocessor determines how many cycles a transfer will take based on its internal information and inputs from the external system.

BLAST# is not valid in the first clock of a bus cycle because the Intel486 DX microprocessor cannot determine the number of cycles a transfer will take until the external system returns KEN#, BS8# and BS16#. BLAST# should only be sampled in the second and subsequent clocks of a cycle when the external system returns RDY# or BRDY#.

The system may terminate a burst cycle by returning RDY# instead of BRDY#. BLAST# will remain deasserted until the last transfer. However, any transfers required to complete a cache line fill will follow the burst order, e.g., if burst order was 4, 0, C, 8 and RDY# was returned at after 0, the next transfers will be from C and 8.

7.2.2.3 Non-Cacheable, Non-Burst, Multiple Cycle Transfers

Figure 7.9 illustrates a 2 cycle non-burst, non-cacheable multiple cycle read. This transfer is simply a sequence of two single cycle transfers. The Intel486 DX microprocessor indicates to the external system that this is a multiple cycle transfer by driving BLAST# inactive during the second clock of the first cycle. The external system returns RDY# active indicating that it will not burst the data. The external system also indicates that the data is not cacheable by returning KEN# inactive one clock before it returns RDY# active. When the Intel486 DX microprocessor samples RDY# active it ignores BRDY#.

Each cycle in the transfer begins when ADS# is driven active and the cycle is complete when the external system returns RDY# active.

The Intel486 DX microprocessor indicates the last cycle of the transfer by driving BLAST# active. The next RDY# returned by the external system terminates the transfer.

7.2.2.4 Non-Cacheable Burst Cycles

The external system converts a multiple cycle request into a burst cycle by returning BRDY# active rather than RDY# in the first cycle of the transfer. This is illustrated in Figure 7.10.

There are several features to note in the burst read. ADS# is only driven active during the first cycle of the transfer. RDY# must be driven inactive when BRDY# is returned active.

BLAST# behaves exactly as it does in the non-burst read. BLAST# is driven inactive in the second clock of the first cycle of the transfer indicating more cycles to follow. In the last cycle, BLAST# is driven active telling the external memory system to end the burst after returning the next BRDY#.

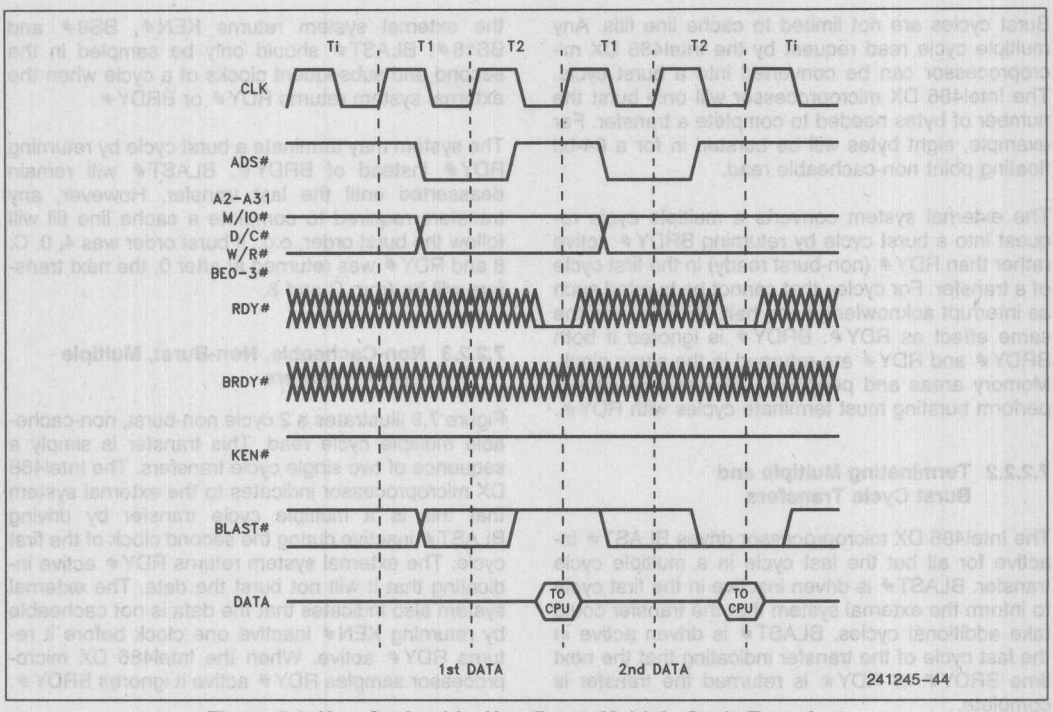


Figure 7.9. Non-Cacheable, Non-Burst, Multiple Cycle Transfers

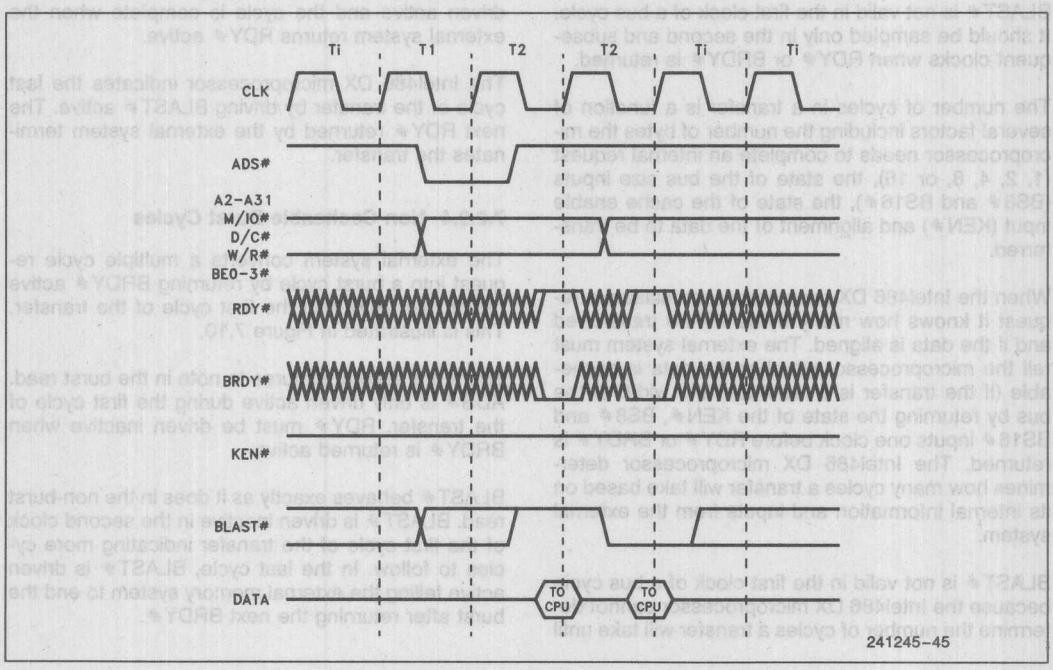


Figure 7.10. Non-Cacheable Burst Cycle

7.2.3 CACHEABLE CYCLES

Any memory read can become a cache fill operation. The external memory system can allow a read request to fill a cache line by returning KEN# active one clock before RDY# or BRDY# during the first cycle of the transfer on the external bus. Once KEN# is asserted and the remaining three requirements described below are met, the Intel486 DX microprocessor will fetch an entire cache line regardless of the state of KEN#. KEN# must be returned active in the last cycle of the transfer for the data to be written into the internal cache. The Intel486 DX microprocessor will only convert memory reads or prefetches into a cache fill.

KEN# is ignored during write or I/O cycles. Memory writes will only be stored in the on-chip cache if there is a cache hit. I/O space is never cached in the internal cache.

To transform a read or a prefetch into a cache line fill the following conditions must be met:

1. The KEN# pin must be asserted one clock prior to RDY# or BRDY# being returned for the first data cycle.
2. The cycle must be of the type that can be internally cached. (Locked reads, I/O reads, and interrupt acknowledge cycles are never cached).
3. The page table entry must have the page cache disable bit (PCD) set to 0. To cache a page table entry, the page directory must have PCD=0. To cache reads or prefetches when paging is disabled, or to cache the page directory entry, control register 3 (CR3) must have PCD=0.
4. The cache disable (CD) bit in control register 0 (CR0) must be clear.

External hardware can determine when the Intel486 DX microprocessor has transformed a read or prefetch into a cache fill by examining the KEN#, M/IO#, D/C#, W/R#, LOCK#, and PCD pins. These pins convey to the system the outcome of conditions 1–3 in the above list. In addition, the Intel486 DX drives PCD high whenever the CD bit in CR0 is set, so that external hardware can evaluate condition 4.

Cacheable cycles can be burst or non-burst.

7.2.3.1 Byte Enables during a Cache Line Fill

For the first cycle in the line fill, the state of the byte enables should be ignored. In a non-cacheable memory read, the byte enables indicate the bytes actually required by the memory or code fetch.

The Intel486 DX microprocessor expects to receive valid data on its entire bus (32 bits) in the first cycle of a cache line fill. Data should be returned with the assumption that all the byte enable pins are driven active. However if BS8# is asserted only one byte need be returned on data lines D0–D7. Similarly if BS16# is asserted two bytes should be returned on D0–D15.

The Intel486 DX microprocessor will generate the addresses and byte enables for all subsequent cycles in the line fill. The order in which data is read during a line fill depends on the address of the first item read. Byte ordering is discussed in Section 7.2.4.

7.2.3.2 Non-Burst Cacheable Cycles

Figure 7.11 shows a non-burst cacheable cycle. The cycle becomes a cache fill when the Intel486 DX microprocessor samples KEN# active at the end of the first clock. The Intel486 DX microprocessor drives BLAST# inactive in the second clock in response to KEN#. BLAST# is driven inactive because a cache fill requires 3 additional cycles to complete. BLAST# remains inactive until the last transfer in the cache line fill. KEN# must be returned active in the last cycle of the transfer for the data to be written into the internal cache.

Note that this cycle would be a single bus cycle if KEN# was not sampled active at the end of the first clock. The subsequent three reads would not have happened since a cache fill was not requested.

The BLAST# output is invalid in the first clock of a cycle. BLAST# may be active during the first clock due to earlier inputs. Ignore BLAST# until the second clock.

During the first cycle of the cache line fill the external system should treat the byte enables as if they are all active. In subsequent cycles in the burst, the Intel486 DX microprocessor drives the address lines and byte enables (see Section 7.2.4.2 for **Burst and Cache Line Fill Order**).

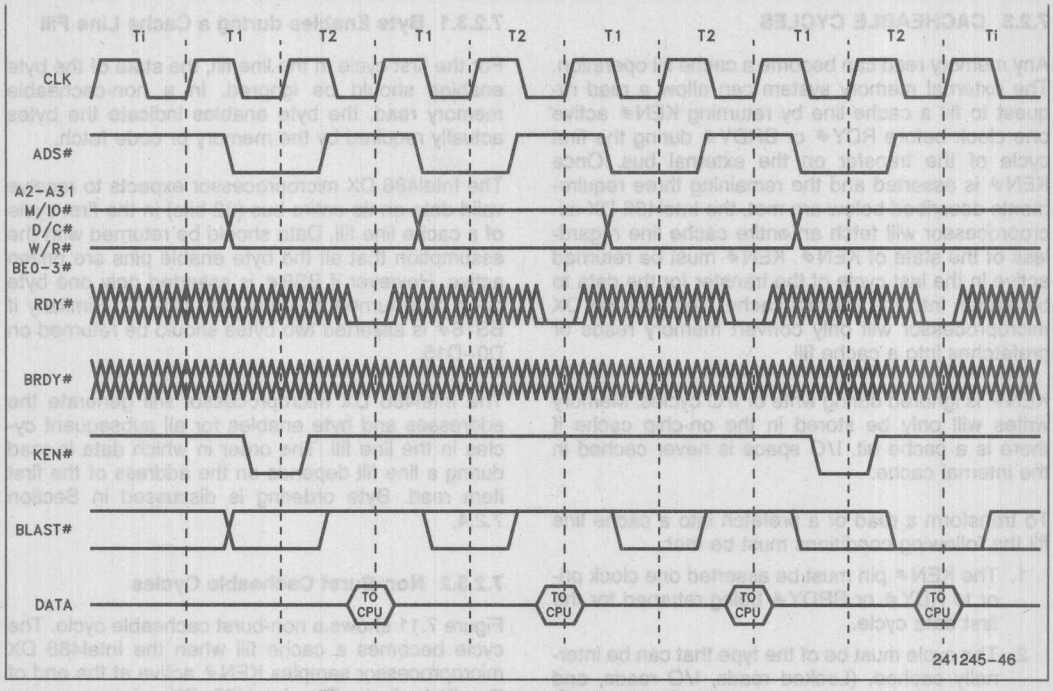


Figure 7.11. Non-Burst, Cacheable Cycles

7.2.3.3 Burst Cacheable Cycles

Figure 7.12 illustrates a burst mode cache fill. As in Figure 7.11, the transfer becomes a cache line fill when the external system returns KEN# active at the end of the first clock in the cycle.

The external system informs the Intel486 DX microprocessor that it will burst the line in by driving BRDY# active at the end of the first cycle in the transfer.

Note that during a burst cycle ADS# is only driven with the first address.

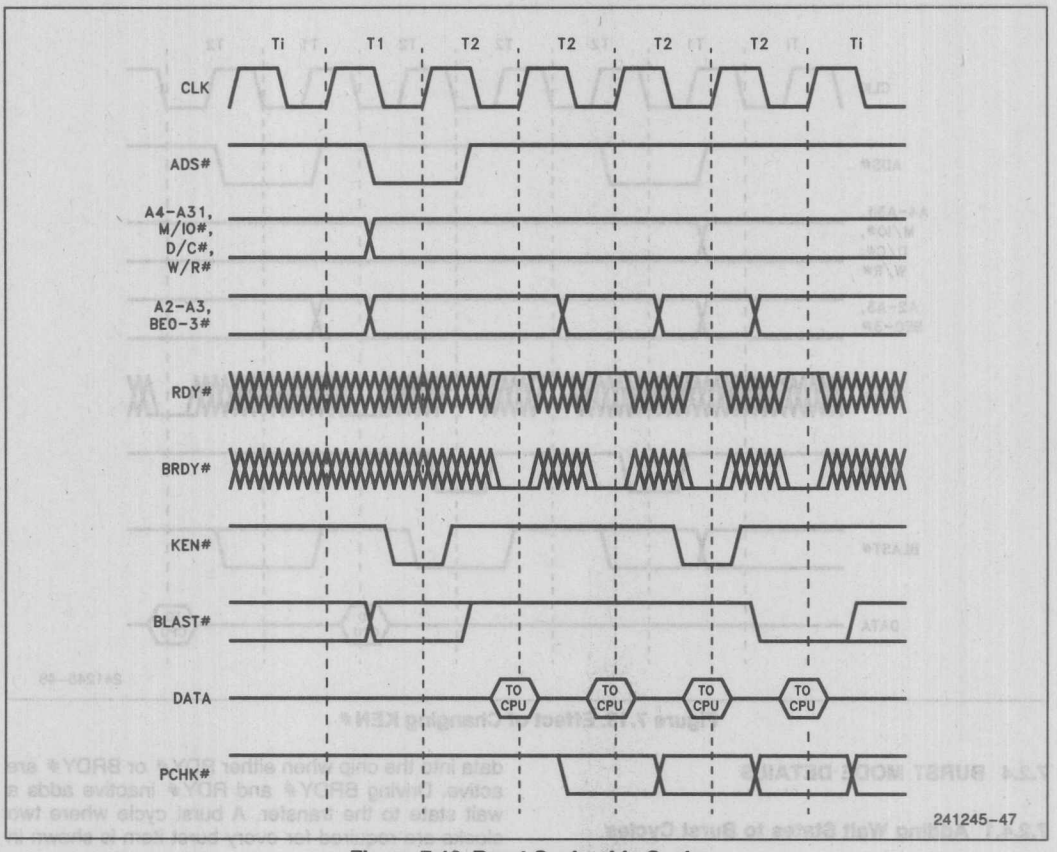


Figure 7.12. Burst Cacheable Cycle

7.2.3.4 Effect of Changing KEN# during a Cache Line Fill

KEN# can change multiple times as long as it arrives at its final value in the clock before RDY# or BRDY# is returned. This is illustrated in Figure 7.13. Note that the timing of BLAST# follows that of KEN# by one clock. The Intel486 DX samples KEN# every clock and uses the value returned in the clock before ready to determine if a bus cycle would be a cache line fill. Similarly, it uses the value of KEN# in the last cycle, before early RDY# to load the line just retrieved from the memory into the cache. KEN# is sampled every clock, it must satisfy setup and hold time.

KEN# can also change multiple times before a burst cycle as long as it arrives at its final value one clock before ready is returned active.

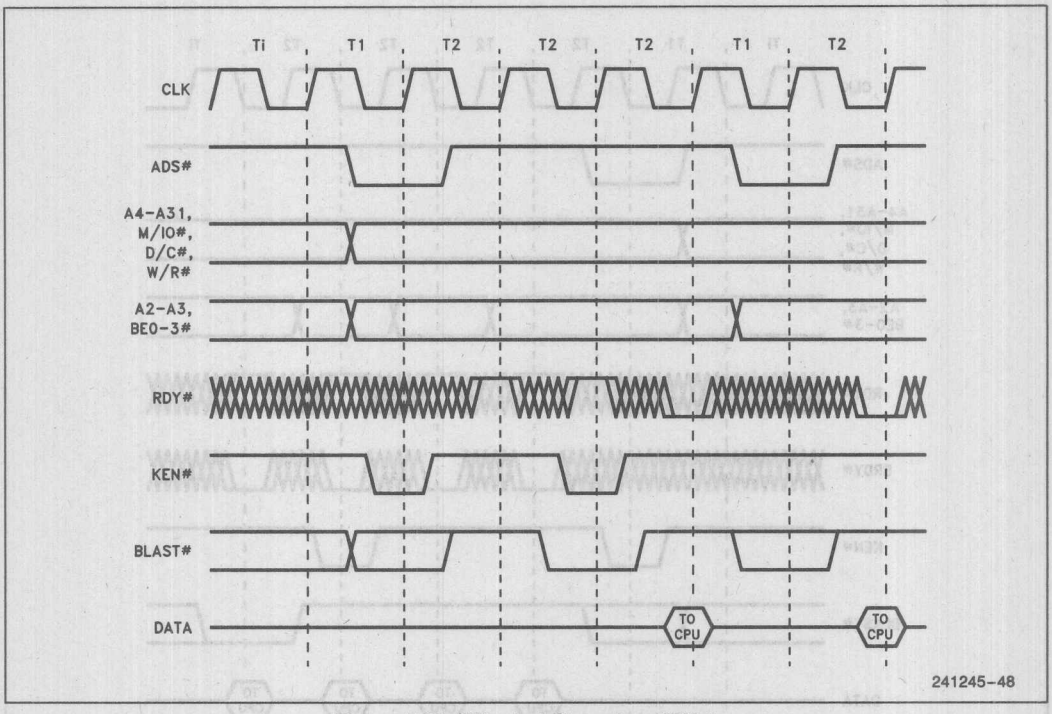


Figure 7.13. Effect of Changing KEN#

7.2.4 BURST MODE DETAILS

7.2.4.1 Adding Wait States to Burst Cycles

Burst cycles need not return data on every clock. The Intel486 DX microprocessor will only strobe data into the chip when either RDY# or BRDY# are active. Driving BRDY# and RDY# inactive adds a wait state to the transfer. A burst cycle where two clocks are required for every burst item is shown in Figure 7.14.

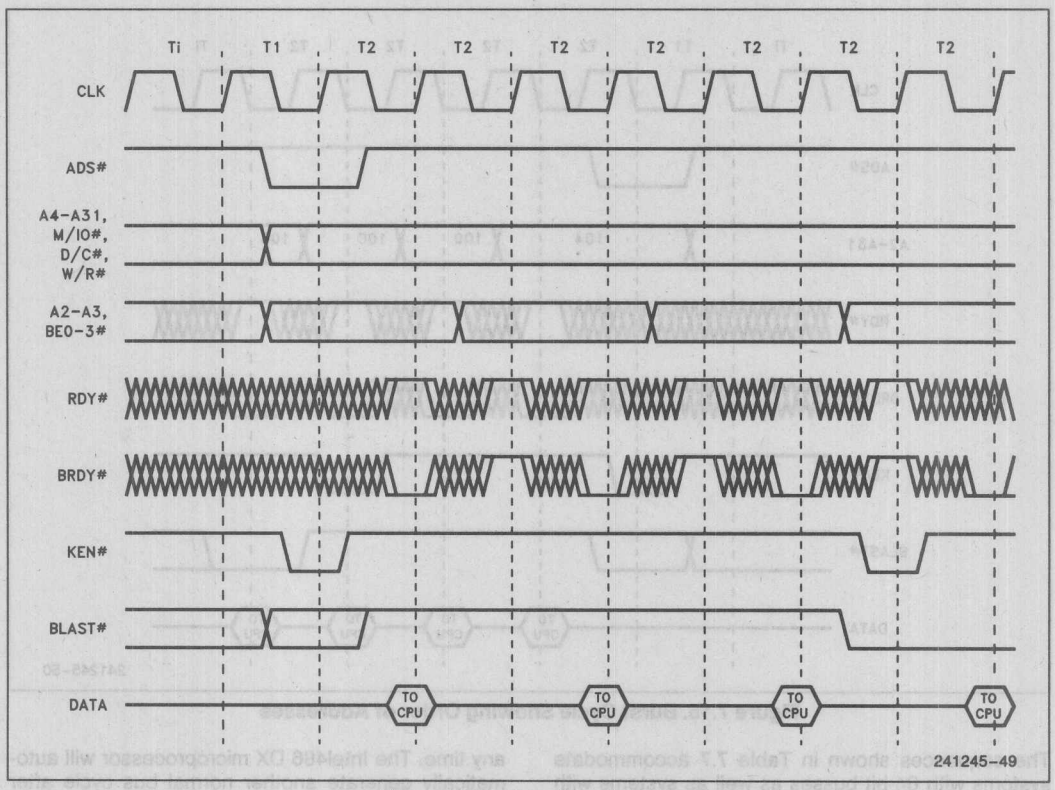


Figure 7.14. Slow Burst Cycle

7.2.4.2 Burst and Cache Line Fill Order

The burst order used by the Intel486 DX microprocessor is shown in Table 7.7. This burst order is followed by any burst cycle (cache or not), cache line fill (burst or not) or code prefetch.

This burst order is optimized for a two-way interleaved memory architecture. This means that if the memory is built as 64-bit (versus 32-bit) words which are multiplexed into the 32-bit data bus, the Intel486 CPU will read all 64 bits before accessing the next location.

The microprocessor presents each request for data in an order determined by the first address in the transfer. For example, if the first address was 104 the next three addresses in the burst will be 100, 10C and 108.

Table 7.7. Burst Order

First Addr.	Second Addr.	Third Addr.	Fourth Addr.
0	4	8	C
4	0	C	8
8	C	0	4
C	8	4	0

An example of burst address sequencing is shown in Figure 7.15.

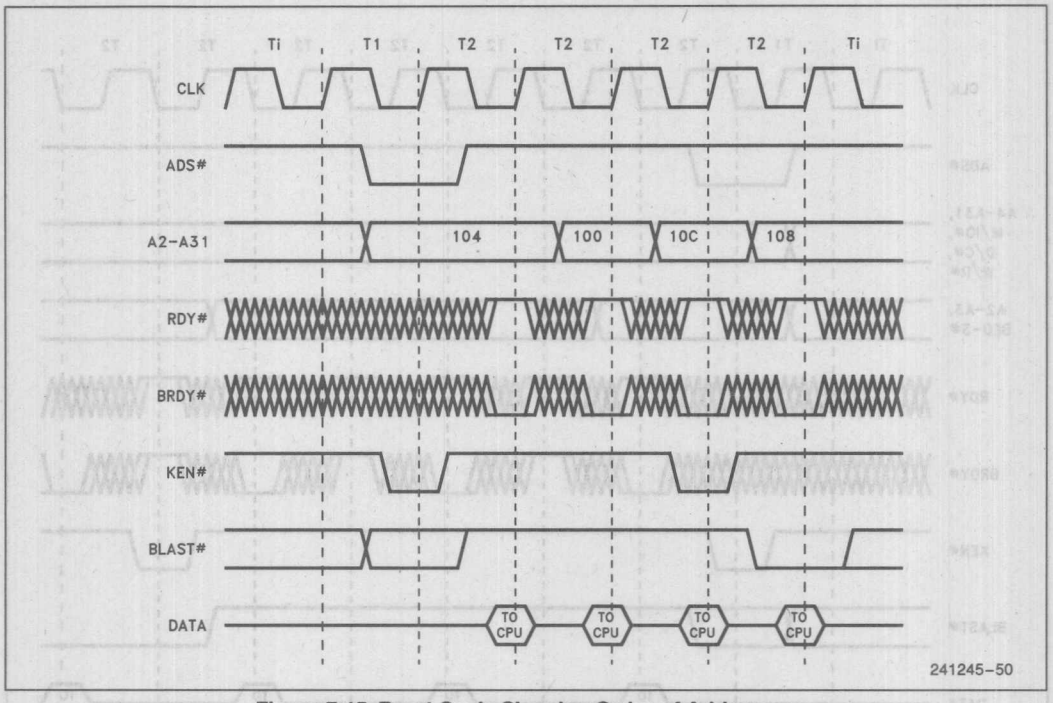


Figure 7.15. Burst Cycle Showing Order of Addresses

The sequences shown in Table 7.7 accommodate systems with 64-bit busses as well as systems with 32-bit data busses. The sequence applies to all bursts, regardless of whether the purpose of the burst is to fill a cache line, do a 64-bit read, or do a pre-fetch. If either BS8# or BS16# is returned active, the Intel486 DX microprocessor completes the transfer of the current 32-bit word before progressing to the next 32-bit word. For example, a BS16# burst to address 4 has the following order: 4-6-0-2-C-E-8-A.

7.2.4.3 Interrupted Burst Cycles

Some memory systems may not be able to respond with burst cycles in the order defined in Table 7.7. To support these systems the Intel486 DX microprocessor allows a burst cycle to be interrupted at any time. The Intel486 DX microprocessor will automatically generate another normal bus cycle after being interrupted to complete the data transfer. This is called an interrupted burst cycle. The external system can respond to an interrupted burst cycle with another burst cycle.

The external system can interrupt a burst cycle by returning RDY# instead of BRDY#. RDY# can be returned after any number of data cycles terminated with BRDY#.

An example of an interrupted burst cycle is shown in Figure 7.16. The Intel486 DX microprocessor immediately drives ADS# active to initiate a new bus cycle after RDY# is returned active. BLAST# is driven inactive one clock after ADS# begins the second bus cycle indicating that the transfer is not complete.

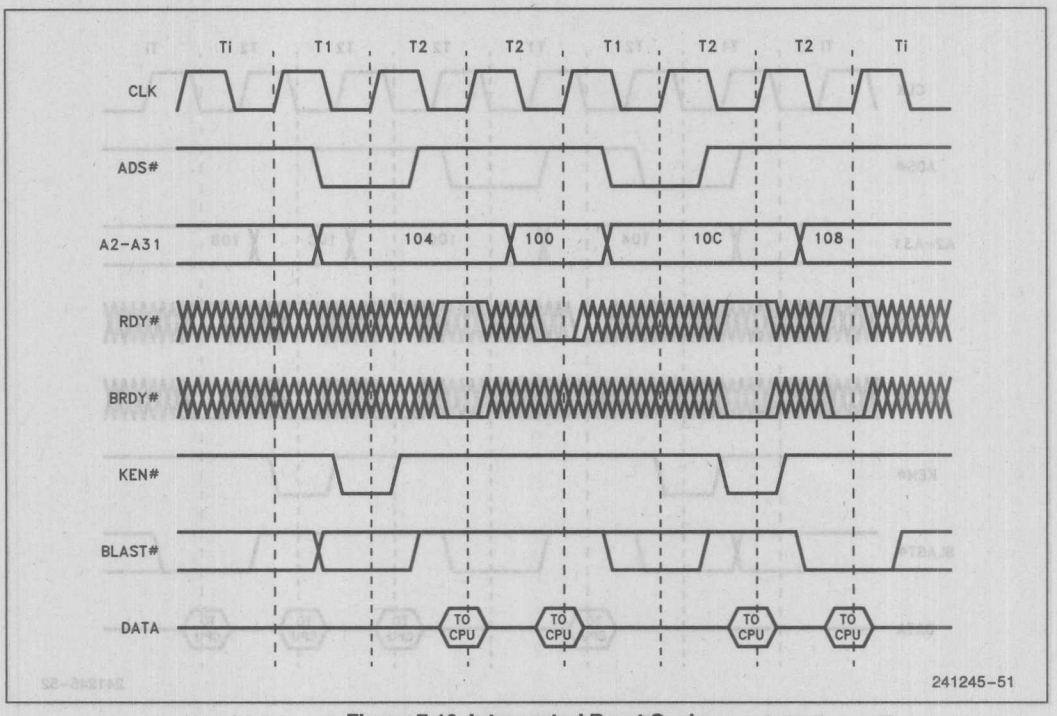


Figure 7.16. Interrupted Burst Cycle

KEN# need not be returned active in the first data cycle of the second part of the transfer in Figure 7.16. The cycle had been converted to a cache fill in the first part of the transfer and the Intel486 DX microprocessor expects the cache fill to be completed. Note that the first half and second half of the transfer in Figure 7.16 are each two cycle burst transfers.

The order in which the Intel486 DX microprocessor requests operands during an interrupted burst transfer is determined by Table 7.7. Mixing RDY# and BRDY# does not change the order in which operand addresses are requested by the Intel486 DX microprocessor.

An example of the order in which the Intel486 DX microprocessor requests operands during a cycle in which the external system mixes RDY# and BRDY# is shown in Figure 7.17. The Intel486 DX microprocessor initially requests a transfer beginning at location 104. The transfer becomes a cache line fill when the external system returns KEN# active. The first cycle of the cache fill transfers the contents of location 104 and is terminated with RDY#. The Intel486 DX microprocessor drives out a new request (by asserting ADS#) to address 100. If the external system terminates the second cycle with BRDY#, the Intel486 DX microprocessor will next request/expect address 10C. The correct order is determined by the first cycle in the transfer, which may not be the first cycle in the burst if the system mixes RDY# with BRDY#.

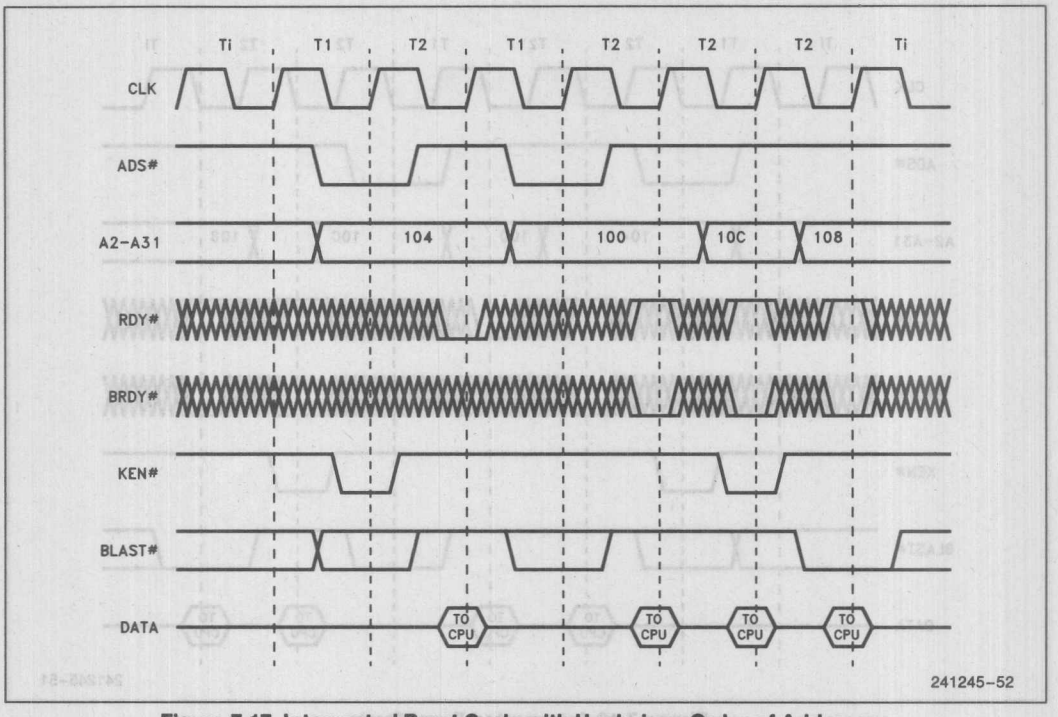


Figure 7.17. Interrupted Burst Cycle with Unobvious Order of Addresses

7.2.5 8- AND 16-BIT CYCLES

The Intel486 DX microprocessor supports both 16- and 8-bit external busses through the BS16# and BS8# inputs. BS16# and BS8# allow the external system to specify, on a cycle by cycle basis, whether the addressed component can supply 8, 16 or 32 bits. BS16# and BS8# can be used in burst cycles as well as non-burst cycles. If both BS16# and BS8# are returned active for any bus cycle, the Intel486 DX microprocessor will respond as if only BS8# were active.

The timing of BS16# and BS8# is the same as that of KEN#. BS16# and BS8# must be driven active before the first RDY# or BRDY# is driven active. Driving the BS16# and BS8# active can force the Intel486 DX microprocessor to run additional cycles to complete what would have been only a single 32-bit cycle. BS8# and BS16# may change the state of BLAST# when they force subsequent cycles from the transfer.

Figure 7.18 shows an example in which BS8# forces the Intel486 DX microprocessor to run two extra cycles to complete a transfer. The Intel486 DX microprocessor issues a request for 24 bits of information. The external system drives BS8# active indicating that only eight bits of data can be supplied per cycle. The Intel486 DX microprocessor issues two extra cycles to complete the transfer.

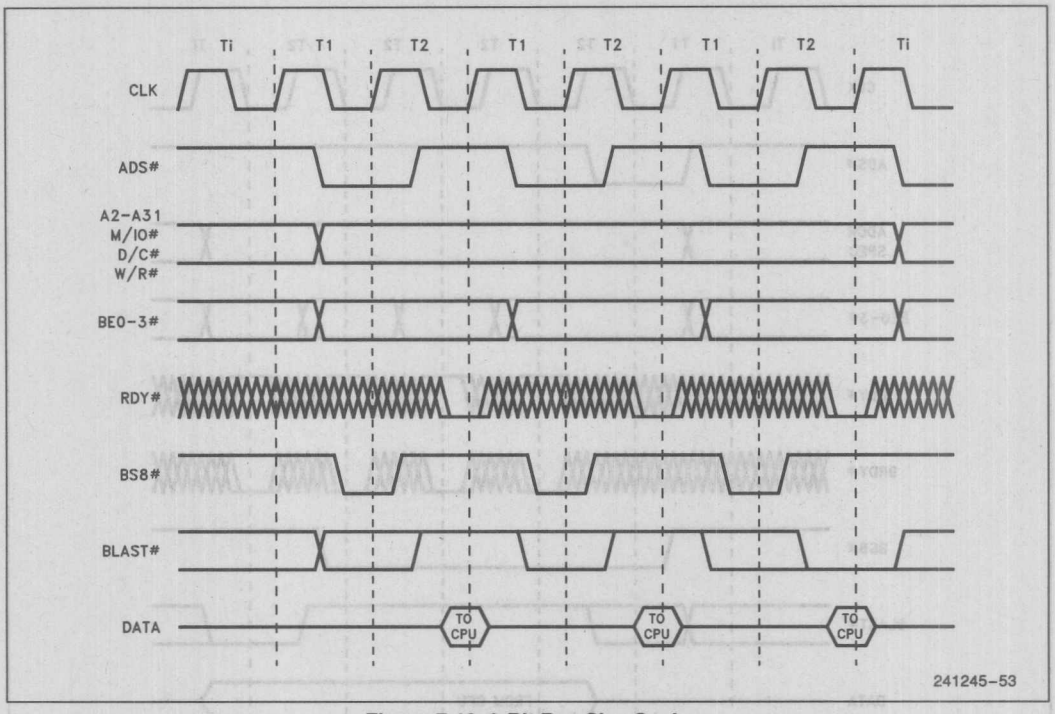


Figure 7.18. 8-Bit Bus Size Cycle

Extra cycles forced by the BS16# and BS8# should be viewed as independent bus cycles. BS16# and BS8# should be driven active for each additional cycle unless the addressed device has the ability to change the number of bytes it can return between cycles. The Intel486 DX microprocessor will drive BLAST# inactive until the last cycle before the transfer is complete.

Refer to Section 7.1.3 for the sequencing of addresses while BS8# or BS16# are active.

BS8# and BS16# operate during burst cycles in exactly the same manner as non-burst cycles. For example, a single non-cacheable read could be transferred by the Intel486 DX microprocessor as four 8-bit burst data cycles. Similarly, a single 32-bit write could be written as four 8-bit burst data cycles. An example of a burst write is shown in Figure 7.19. Burst writes can only occur if BS8# or BS16# is asserted.

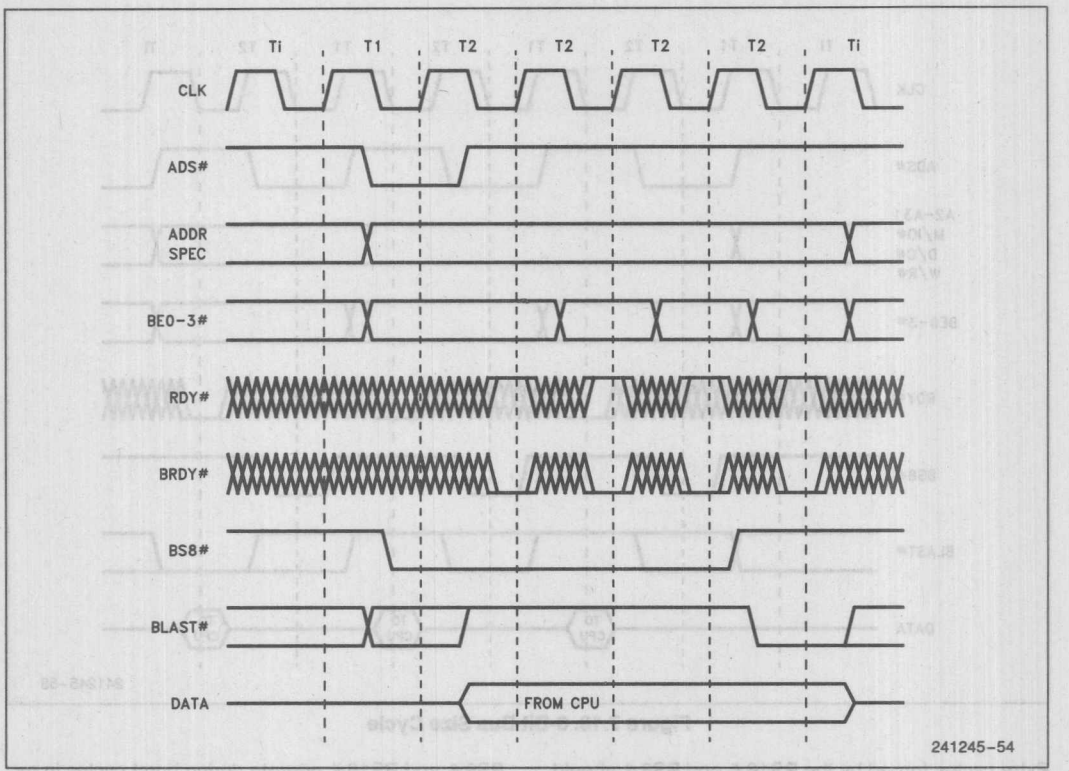


Figure 7.19. Burst Write as a Result of BS8# or BS16#

7.2.6 LOCKED CYCLES

Locked cycles are generated in software for any instruction that performs a read-modify-write operation. During a read-modify-write operation the processor can read and modify a variable in external memory and be assured that the variable is not accessed between the read and write.

Locked cycles are automatically generated during certain bus transfers. The xchg (exchange) instruction generates a locked cycle when one of its operands is memory based. Locked cycles are generated when a segment or page table entry is updated and during interrupt acknowledge cycles. Locked cycles are also generated when the LOCK instruction prefix is used with selected instructions.

Locked cycles are implemented in hardware with the LOCK# pin. When LOCK# is active, the processor is performing a read-modify-write operation and the external bus should not be relinquished until the cycle is complete. Multiple reads or writes can be locked. A locked cycle is shown in Figure 7.20. LOCK# goes active with the address and bus definition pins at the beginning of the first read cycle and remains active until RDY# is returned for the last write cycle. For unaligned 32 bits read-modify-write operation, the LOCK# remains active for the entire duration of the multiple cycle. It will go inactive when RDY# is returned for the last write cycle.

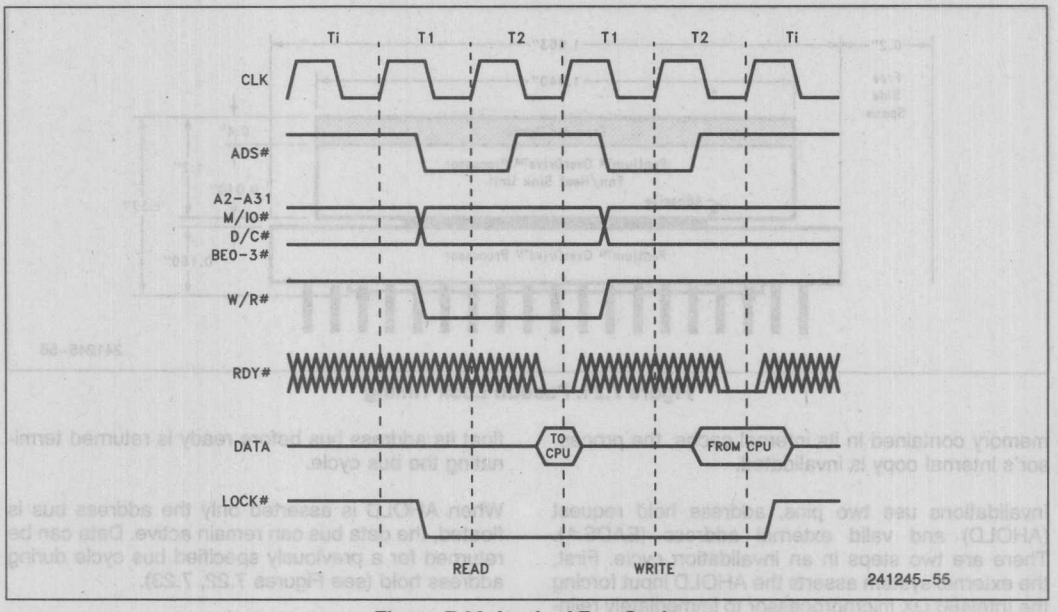


Figure 7.20. Locked Bus Cycle

When LOCK# is active, the Intel486 DX microprocessor will recognize address hold and backoff but will not recognize bus hold. It is left to the external system to properly arbitrate a central bus when the Intel486 DX microprocessor generates LOCK#.

7.2.7 PSEUDO-LOCKED CYCLES

Pseudo-locked cycles assure that no other master will be given control of the bus during operand transfers which take more than one bus cycle. Examples include 64-bit floating point read and writes, 64-bit descriptor loads and cache line fills.

Pseudo-locked transfers are indicated by the PLOCK# pin. The memory operands must be aligned for correct operation of a pseudo-locked cycle.

PLOCK# need not be examined during burst reads. A 64-bit aligned operand can be retrieved in one burst (note: this is only valid in systems that do not interrupt bursts).

The system must examine PLOCK# during 64-bit writes since the Intel486 DX microprocessor cannot burst write more than 32 bits. However, burst can be used within each 32-bit write cycle if BS8# or BS16# is asserted. BLAST will be deasserted in response to BS8# or BS16#. A 64-bit write will be driven out as two non-burst bus cycles. BLAST# is asserted during both writes since a burst is not possible.

PLOCK# is asserted during the first write to indicate that another write follows. This behavior is shown in Figure 7.21.

The first cycle of a 64-bit floating point write is the only case in which both PLOCK# and BLAST# are asserted. Normally PLOCK# and BLAST# are the inverse of each other.

During all of the cycles where PLOCK# is asserted, HOLD is not acknowledged until the cycle completes. This results in a large HOLD latency, especially when BS8# or BS16# is asserted. To reduce the HOLD latency during these cycles, windows are available between transfers to allow HOLD to be acknowledged during non-cacheable, non-bursted code prefetches. PLOCK# will be asserted since BLAST# is negated, but it is ignored and HOLD is recognized during the prefetch.

PLOCK# can change several times during a cycle settling to its final value in the clock ready is returned.

7.2.8 INVALIDATE CYCLES

Invalidate cycles are needed to keep the Intel486 DX microprocessor's internal cache contents consistent with external memory. The Intel486 DX microprocessor contains a mechanism for listening to writes by other devices to external memory. When the processor finds a write to a Section of external

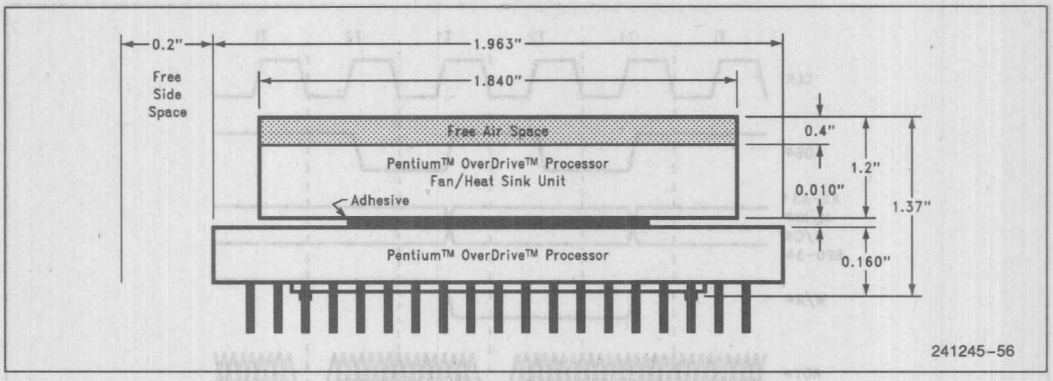


Figure 7.21. Pseudo Lock Timing

memory contained in its internal cache, the processor's internal copy is invalidated.

Invalidations use two pins, address hold request (AHOLD) and valid external address (EADS#). There are two steps in an invalidation cycle. First, the external system asserts the AHOLD input forcing the Intel486 DX microprocessor to immediately relinquish its address bus. Next, the external system asserts EADS# indicating that a valid address is on the Intel486 DX microprocessor's address bus. EADS# and the invalidation address, Figure 7-22 shows the fastest possible invalidation cycle. The Intel486 DX CPU recognizes AHOLD on one CLK edge and floats the address bus in response. To allow the address bus to float and avoid contention, EADS# and the invalidation address should not be driven until the following CLK edge. The microprocessor reads the address over its address lines. If the microprocessor finds this address in its internal cache, the cache entry is invalidated. Note that the Intel486 DX microprocessor's address bus is input/output unlike the Intel386 microprocessor's bus, which is output only.

The Intel486 DX microprocessor immediately relinquishes its address bus in the next clock upon assertion of AHOLD. For example, the bus could be 3 wait states into a read cycle. If AHOLD is activated, the Intel486 DX microprocessor will immediately float its address bus before ready is returned terminating the bus cycle.

When AHOLD is asserted only the address bus is floated, the data bus can remain active. Data can be returned for a previously specified bus cycle during address hold (see Figures 7.22, 7.23).

EADS# is normally asserted when an external master drives an address onto the bus. AHOLD need not be driven for EADS# to generate an internal invalidate. If EADS# alone is asserted while the Intel486 DX microprocessor is driving the address bus, it is possible that the invalidation address will come from the Intel486 DX microprocessor itself.

Note that it is also possible to run an invalidation cycle by asserting EADS# when HOLD or BOFF# is asserted.

Running an invalidate cycle prevents the Intel486 DX microprocessor cache from satisfying other internal requests, so invalidations should be run only when necessary. The fastest possible invalidate cycle is shown in Figure 7.22, while a more realistic invalidation cycle is shown in 7.23. Both of the examples take one clock of cache access from the rest of the Intel486 DX microprocessor.

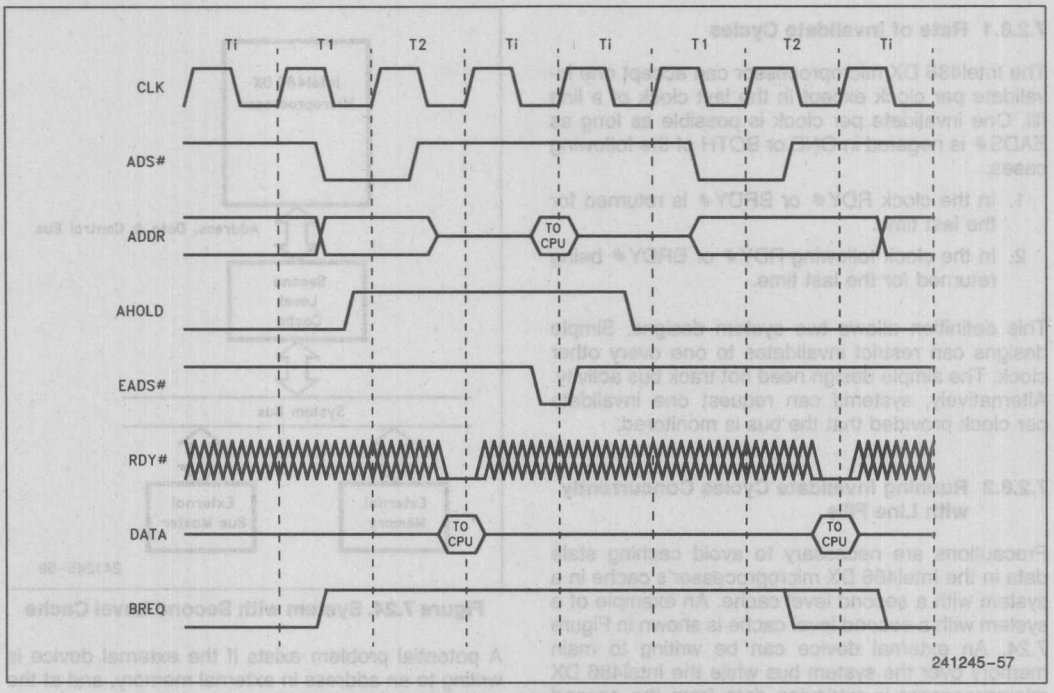


Figure 7.22. Fast Internal Cache Invalidation Cycle

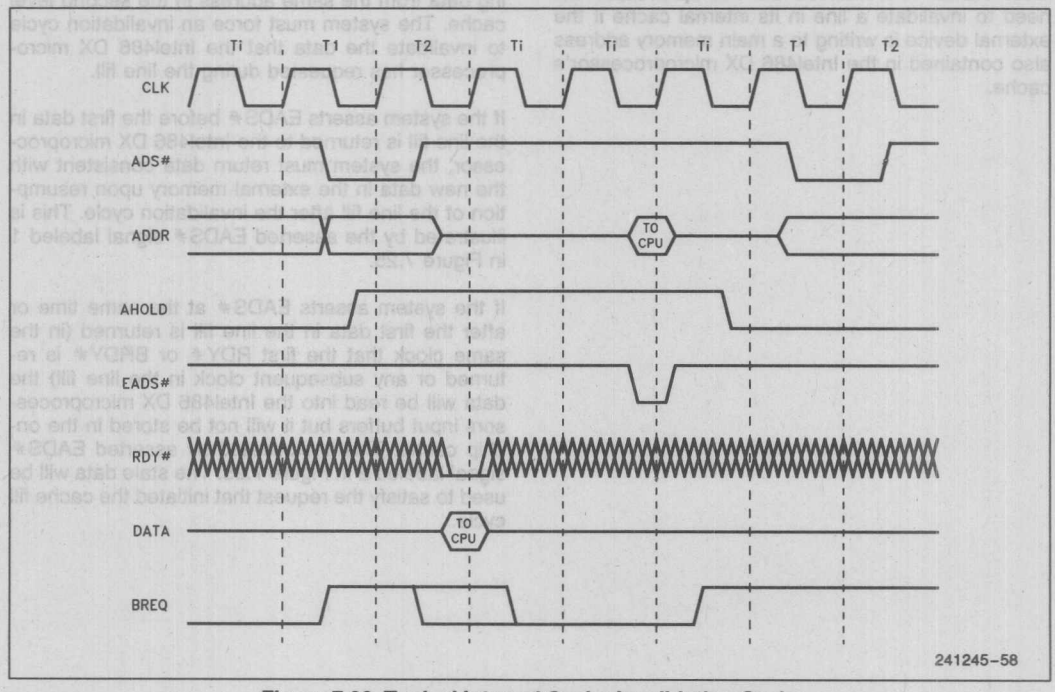


Figure 7.23. Typical Internal Cache Invalidation Cycle

7.2.8.1 Rate of Invalidate Cycles

The Intel486 DX microprocessor can accept one invalidate per clock except in the last clock of a line fill. One invalidate per clock is possible as long as EADS# is negated in ONE or BOTH of the following cases:

1. In the clock RDY# or BRDY# is returned for the last time.
2. In the clock following RDY# or BRDY# being returned for the last time.

This definition allows two system designs. Simple designs can restrict invalidates to one every other clock. The simple design need not track bus activity. Alternatively, systems can request one invalidate per clock provided that the bus is monitored.

7.2.8.2 Running Invalidate Cycles Concurrently with Line Fills

Precautions are necessary to avoid caching stale data in the Intel486 DX microprocessor's cache in a system with a second level cache. An example of a system with a second level cache is shown in Figure 7.24. An external device can be writing to main memory over the system bus while the Intel486 DX microprocessor is retrieving data from the second level cache. The Intel486 DX microprocessor will need to invalidate a line in its internal cache if the external device is writing to a main memory address also contained in the Intel486 DX microprocessor's cache.

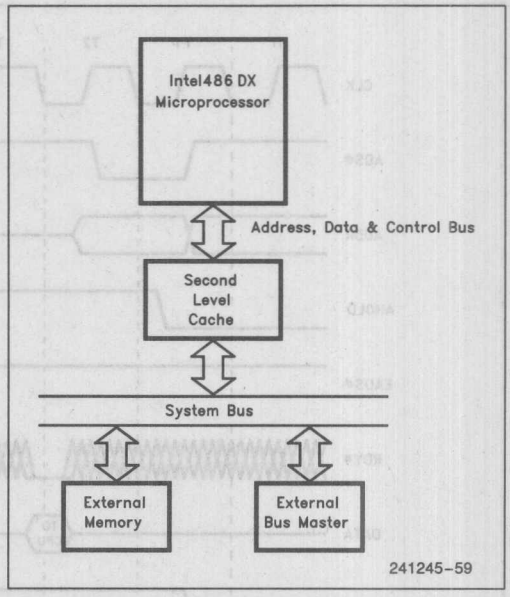
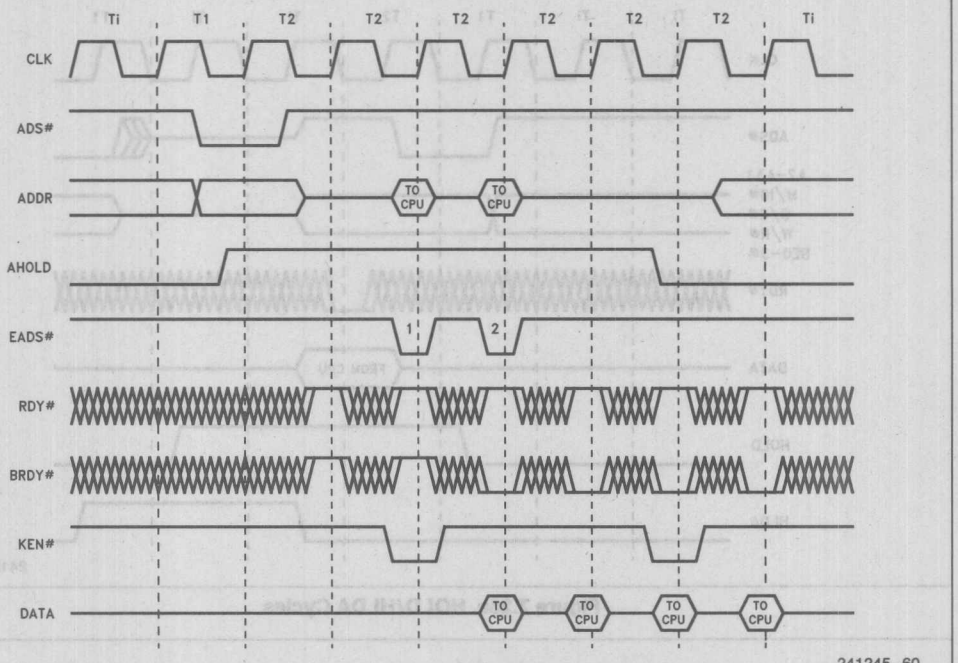


Figure 7.24. System with Second Level Cache

A potential problem exists if the external device is writing to an address in external memory, and at the same time the Intel486 DX microprocessor is reading data from the same address in the second level cache. The system must force an invalidation cycle to invalidate the data that the Intel486 DX microprocessor has requested during the line fill.

If the system asserts EADS# before the first data in the line fill is returned to the Intel486 DX microprocessor, the system must return data consistent with the new data in the external memory upon resumption of the line fill after the invalidation cycle. This is illustrated by the asserted EADS# signal labeled 1 in Figure 7.25.

If the system asserts EADS# at the same time or after the first data in the line fill is returned (in the same clock that the first RDY# or BRDY# is returned or any subsequent clock in the line fill) the data will be read into the Intel486 DX microprocessors input buffers but it will not be stored in the on-chip cache. This is illustrated by asserted EADS# signal labeled 2 in Figure 7.25. The stale data will be used to satisfy the request that initiated the cache fill cycle.



NOTES:
1. Data returned must be consistent if its address equals the invalidation address in this clock
2. Data returned will not be cached if its address equals the invalidation address in this clock

Figure 7.25. Cache Invalidation Cycle Concurrent with Line Fill

7.2.9 BUS HOLD

The Intel486 DX microprocessor provides a bus hold, hold acknowledge protocol using the bus hold request (HOLD) and bus hold acknowledge (HLDA) pins. Asserting the HOLD input indicates that another bus master desires control of the Intel486 DX microprocessor's bus. The processor will respond by floating its bus and driving HLDA active when the current bus cycle, or sequence of locked cycles is complete. An example of a HOLD/HLDA transaction is shown in Figure 7.26a. Unlike the Intel386 microprocessor, the Intel486 DX microprocessor can respond to HOLD by floating its bus and asserting HLDA while RESET is asserted.

Note that HOLD will be recognized during un-aligned writes (less than or equal to 32-bits) with BLAST# being active for each write. For greater than 32-bit or un-aligned write, HOLD# recognition is prevented by PLOCK# getting asserted.

For cacheable and nonbursted or bursted cycles, HOLD is acknowledged during backoff only if HOLD and BOFF# are asserted during an active bus cycle (after ADS# asserted) and before the first RDY# or BRDY# has been returned (see Figure 7.26b). The order in which HOLD and BOFF# go active is unimportant (so long as both are active prior to the first RDY#/BRDY# returned by the system). Figure 7.26b shows the case where HOLD is asserted first; HOLD could be asserted simultaneously or after BOFF# and still be acknowledged.

The pins floated during bus hold are: BE0#–BE3#, PCD, PWT, W/R#, D/C#, M/IO#, LOCK#, PLOCK#, ADS#, BLAST#, D0–D31, A2–A31, DP0–DP3.

7.2.10 INTERRUPT ACKNOWLEDGE

The Intel486 DX microprocessor generates interrupt acknowledge cycles in response to maskable interrupt requests generated on the interrupt request input (INTR) pin. Interrupt acknowledge cycles have a unique cycle type generated on the cycle type pins.

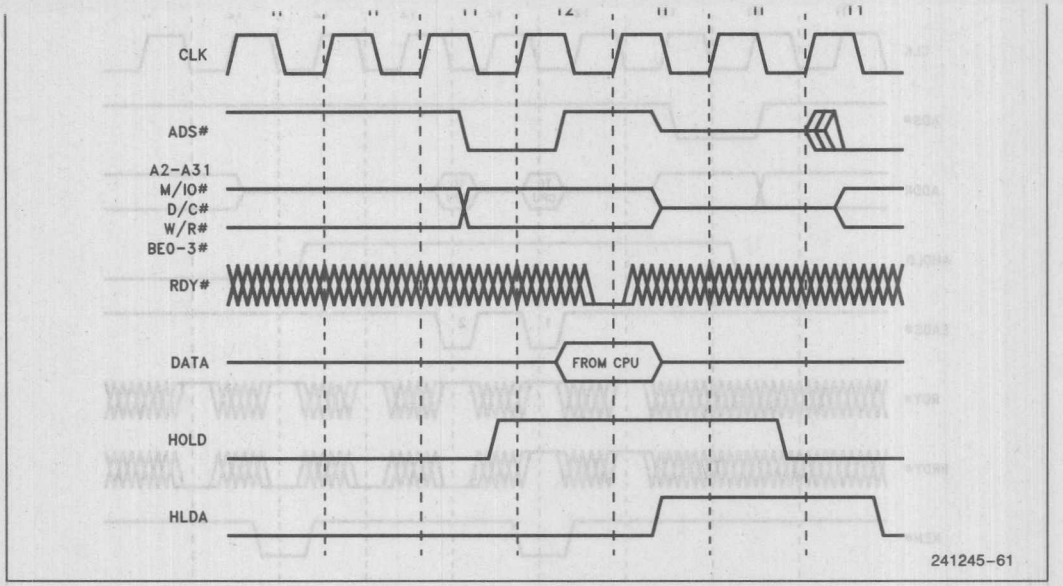


Figure 7.26a. HOLD/HLDA Cycles

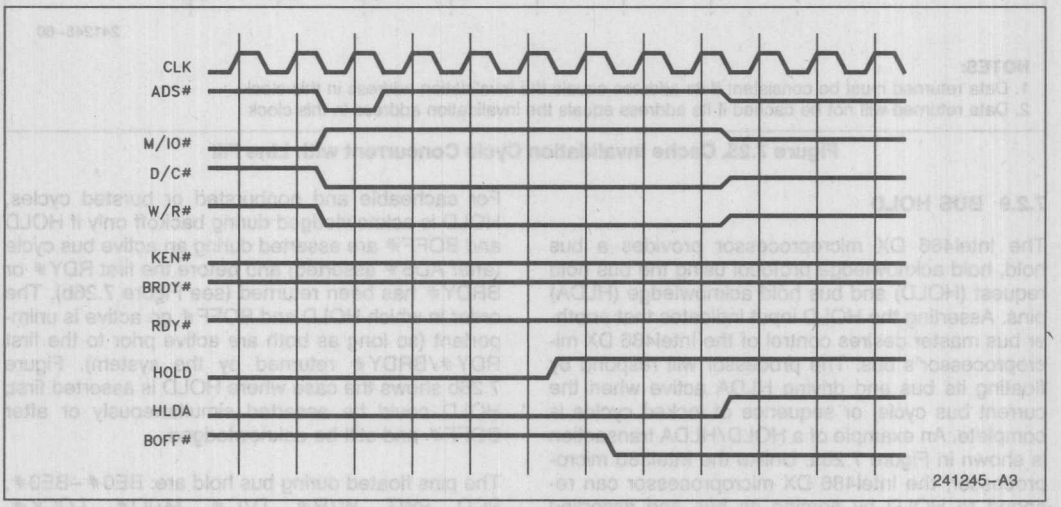


Figure 7.26b. HOLD Request Acknowledge During BOFF#

An example interrupt acknowledge transaction is shown in Figure 7.27. Interrupt acknowledge cycles are generated in locked pairs. Data returned during the first cycle is ignored. The interrupt vector is returned during the second cycle on the lower 8 bits of the data bus. The Intel486 DX microprocessor has 256 possible interrupt vectors.

The state of A2 distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A31–A3 low, A2 high, BE3#–BE1# high, and BE0# low). The address driven during the second interrupt acknowledge cycle is 0 (A31–A2 low, BE3#–BE1# high, BE0# low).

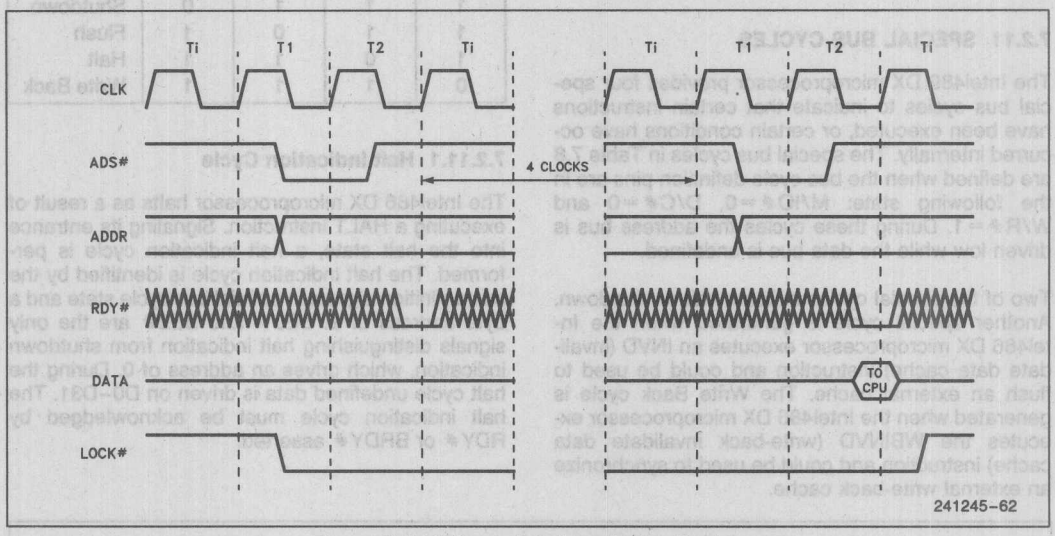


Figure 7.27. Interrupt Acknowledge Cycles

Each of the interrupt acknowledge cycles are terminated when the external system returns RDY# or BRDY#. Wait states can be added by withholding RDY# or BRDY#. The Intel486 DX microprocessor automatically generates four idle clocks between the first and second cycles to allow for 8259A recovery time.

7.2.11 SPECIAL BUS CYCLES

The Intel486 DX microprocessor provides four special bus cycles to indicate that certain instructions have been executed, or certain conditions have occurred internally. The special bus cycles in Table 7.8 are defined when the bus cycle definition pins are in the following state: M/IO# = 0, D/C# = 0 and W/R# = 1. During these cycles the address bus is driven low while the data bus is undefined.

Two of the special cycles indicate halt or shutdown. Another special cycle is generated when the Intel486 DX microprocessor executes an INVD (invalidate data cache) instruction and could be used to flush an external cache. The Write Back cycle is generated when the Intel486 DX microprocessor executes the WBINVD (write-back invalidate data cache) instruction and could be used to synchronize an external write-back cache.

The external hardware must acknowledge these special bus cycles by returning RDY# or BRDY#.

Table 7.8. Special Bus Cycle Encoding

BE3#	BE2#	BE1#	BE0#	Special Bus Cycle
1	1	1	0	Shutdown
1	1	0	1	Flush
1	0	1	1	Halt
0	1	1	1	Write Back

7.2.11.1 Halt Indication Cycle

The Intel486 DX microprocessor halts as a result of executing a HALT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the bus definition signals in special bus cycle state and a byte address of 2. BE0# and BE2# are the only signals distinguishing halt indication from shutdown indication, which drives an address of 0. During the halt cycle undefined data is driven on D0–D31. The halt indication cycle must be acknowledged by RDY# or BRDY# asserted.

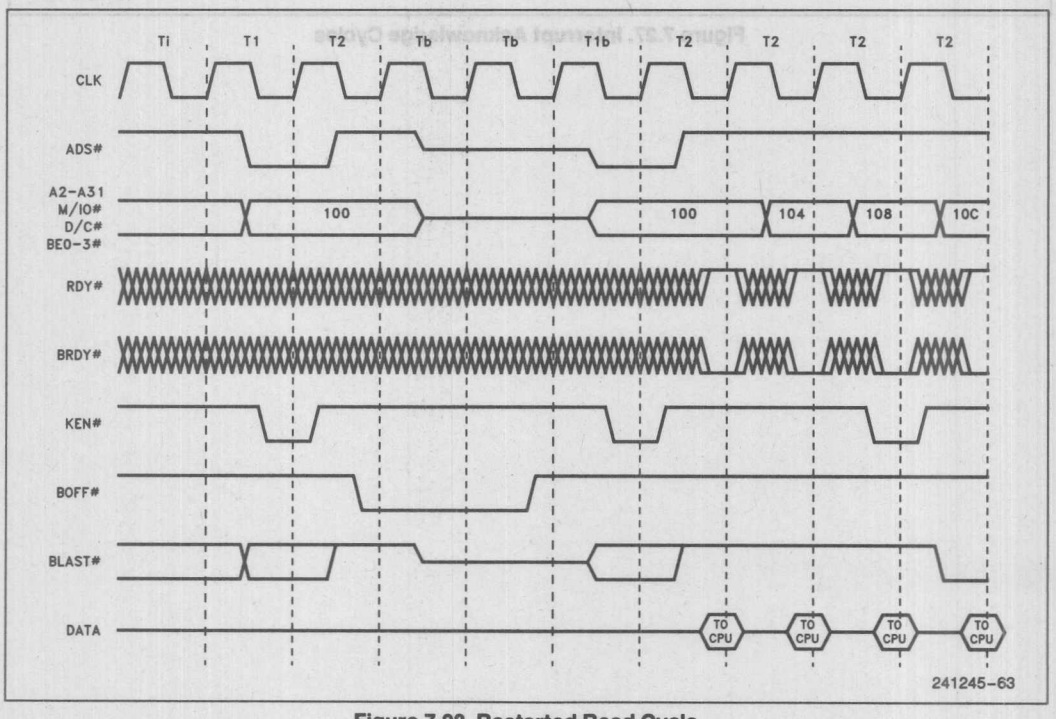


Figure 7.28. Restarted Read Cycle

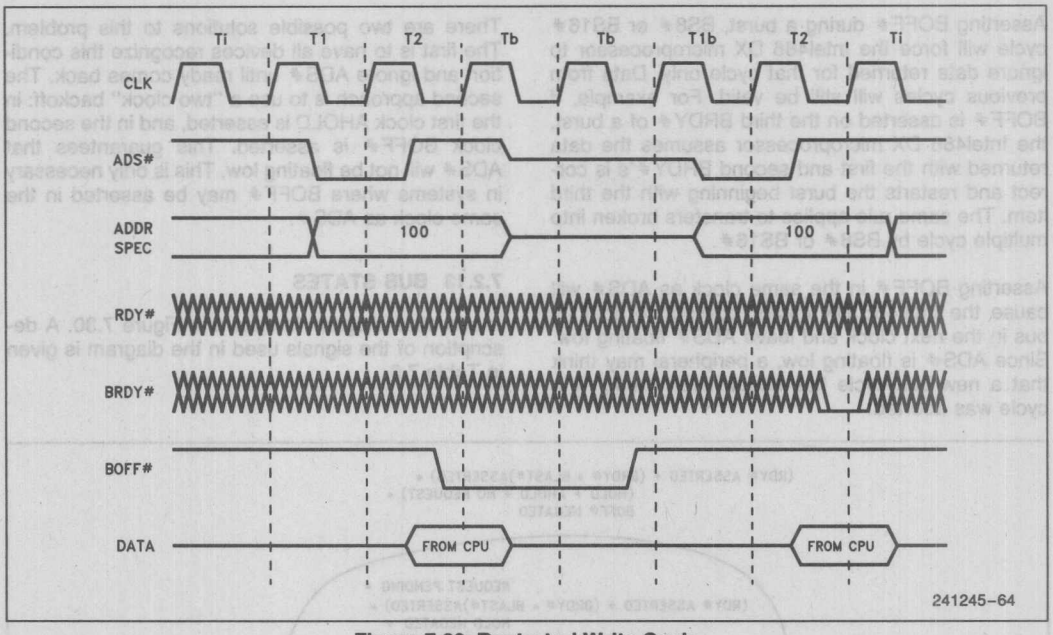


Figure 7.29. Restarted Write Cycle

A halted Intel486 DX microprocessor resumes execution when INTR (if interrupts are enabled) or NMI or RESET is asserted.

7.2.11.2 Shutdown Indication Cycle

The Intel486 DX microprocessor shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the bus definition signals in special bus cycle state and a byte address of 0.

7.2.12 BUS CYCLE RESTART

In a multi-master system another bus master may require the use of the bus to enable the Intel486 DX microprocessor to complete its current bus request. In this situation the Intel486 DX microprocessor will need to restart its bus cycle after the other bus master has completed its bus transaction.

A bus cycle may be restarted if the external system asserts the backoff (BOFF#) input. The Intel486 DX microprocessor samples the BOFF# pin every clock. The Intel486 DX microprocessor will immediately (in the next clock) float its address, data and status pins when BOFF# is asserted (see Figure 7.28). Any bus cycle in progress when BOFF# is asserted is aborted and any data returned to the processor is ignored. The same pins are floated in response to BOFF# as are floated in response to HOLD. HLDA is not generated in response to BOFF#. BOFF# has higher priority than RDY# or BRDY#. If either RDY# or BRDY# are returned in the same clock as BOFF#, BOFF# takes effect.

The device asserting BOFF# is free to run any cycles it wants while the Intel486 DX microprocessor bus is in its high impedance state. If backoff is requested after the Intel486 DX microprocessor has started a cycle, the new master should wait for memory to return RDY# or BRDY# before assuming control of the bus. Waiting for ready provides a handshake to insure that the memory system is ready to accept a new cycle. If the bus is idle when BOFF# is asserted, the new master can start its cycle two clocks after issuing BOFF#.

The external memory can view BOFF# in the same manner as BLAST#. Asserting BOFF# tells the external memory system that the current cycle is the last cycle in a transfer.

The bus remains in the high impedance state until BOFF# is negated. Upon negation, the Intel486 DX microprocessor restarts its bus cycle by driving out the address and status and asserting ADS#. The bus cycle then continues as usual.

Asserting BOFF# during a burst, BS8# or BS16# cycle will force the Intel486 DX microprocessor to ignore data returned for that cycle only. Data from previous cycles will still be valid. For example, if BOFF# is asserted on the third BRDY# of a burst, the Intel486 DX microprocessor assumes the data returned with the first and second BRDY#'s is correct and restarts the burst beginning with the third item. The same rule applies to transfers broken into multiple cycle by BS8# or BS16#.

Asserting BOFF# in the same clock as ADS# will cause the Intel486 DX microprocessor to float its bus in the next clock and leave ADS# floating low. Since ADS# is floating low, a peripheral may think that a new bus cycle has begun even though the cycle was aborted.

There are two possible solutions to this problem. The first is to have all devices recognize this condition and ignore ADS# until ready comes back. The second approach is to use a "two clock" backoff: in the first clock AHOLD is asserted, and in the second clock BOFF# is asserted. This guarantees that ADS# will not be floating low. This is only necessary in systems where BOFF# may be asserted in the same clock as ADS#.

7.2.13 BUS STATES

A bus state diagram is shown in Figure 7.30. A description of the signals used in the diagram is given in Table 7.9.

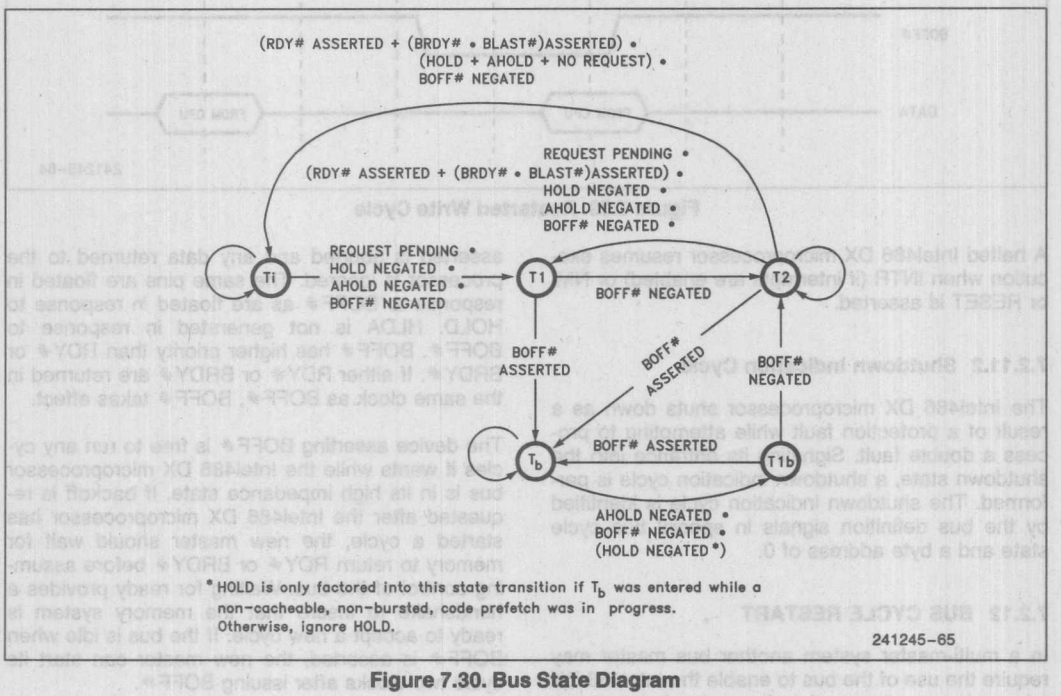


Figure 7.30. Bus State Diagram

Table 7.9. Bus State Description

State	Means
Ti	Bus is idle. Address and status signals may be driven to undefined values, or the bus may be floated to a high impedance state.
T1	First clock cycle of a bus cycle. Valid address and status are driven and ADS# is asserted.
T2	Second and subsequent clock cycles of a bus cycle. Data is driven if the cycle is a write, or data is expected if the cycle is a read. RDY# and BRDY# are sampled.
T1b	First clock cycle of a restarted bus cycle. Valid address and status are driven and ADS# is asserted.
Tb	Second and subsequent clock cycles of an aborted bus cycle.

7.2.14 FLOATING POINT ERROR HANDLING

The Intel486 DX microprocessor provides two options for reporting floating point errors. The simplest method is to raise interrupt 16 whenever an unmasked floating point error occurs. This option may be enabled by setting the NE bit in control register 0 (CR0).

The Intel486 DX microprocessor also provides the option of allowing external hardware to determine how floating point errors are reported. This option is necessary for compatibility with the error reporting scheme used in DOS based systems. The NE bit must be cleared in CR0 to enable user-defined error reporting. User-defined error reporting is the default condition because the NE bit is cleared on reset.

Two pins, floating point error (FERR#) and ignore numeric error (IGNNE#), are provided to direct the actions of hardware if user-defined error reporting is used. The Intel486 DX microprocessor asserts the FERR# output to indicate that a floating point error has occurred. FERR# corresponds to the ERROR# pin on the Intel387 math coprocessor. However, there is a difference in the behavior of the two.

In some cases FERR# is asserted when the next floating point instruction is encountered and in other cases it is asserted before the next floating point instruction is encountered depending upon the execution state of the instruction causing the exception.

The following class of floating point exceptions drive FERR# at the time the exception occurs (i.e., before encountering the next floating point instruction).

1. The stack fault, invalid operation, and denormal exceptions on all transcendental instructions, integer arithmetic instructions, FSQRT, FSEALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exceptions on store instructions (including integer store instructions).

The following class of floating point exceptions drive FERR# only after encountering the next floating point instruction.

1. Exceptions other than on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exception on all basic arithmetic, load, compare, and control instructions (i.e., all other instructions).

For both sets of exceptions above, the Intel387 Math Coprocessor asserts ERROR# when the error occurs and does not wait for the next floating point instruction to be encountered.

IGNNE# is an input to the Intel486 DX microprocessor.

When the NE bit in CR0 is cleared, and IGNNE# is asserted, the Intel486 DX microprocessor will ignore a user floating point error and continue executing floating point instructions. When IGNNE# is negated, the Intel486 DX microprocessor will freeze on floating point instructions which get errors (except for the control instructions FNCLEX, FNINIT, FNSAVE, FNSTENV, FNSTCW, FNSTSW, FNSTSW AX, FNENI, FNDISI and FNSETPM). IGNNE# may be asynchronous to the Intel486 DX clock.

In systems with user-defined error reporting, the FERR# pin is connected to the interrupt controller. When an unmasked floating point error occurs, an interrupt is raised. If IGNNE# is high at the time of this interrupt, the Intel486 DX microprocessor will freeze (disallowing execution of a subsequent floating point instruction) until the interrupt handler is invoked. By driving the IGNNE# pin low (when clearing the interrupt request), the interrupt handler can allow execution of a floating point instruction, within the interrupt handler, before the error condition is cleared (by FNCLEX, FNINIT, FNSAVE or FNSTENV). If execution of a non-control floating point instruction, within the floating point interrupt handler, is not needed, the IGNNE# pin can be tied HIGH.

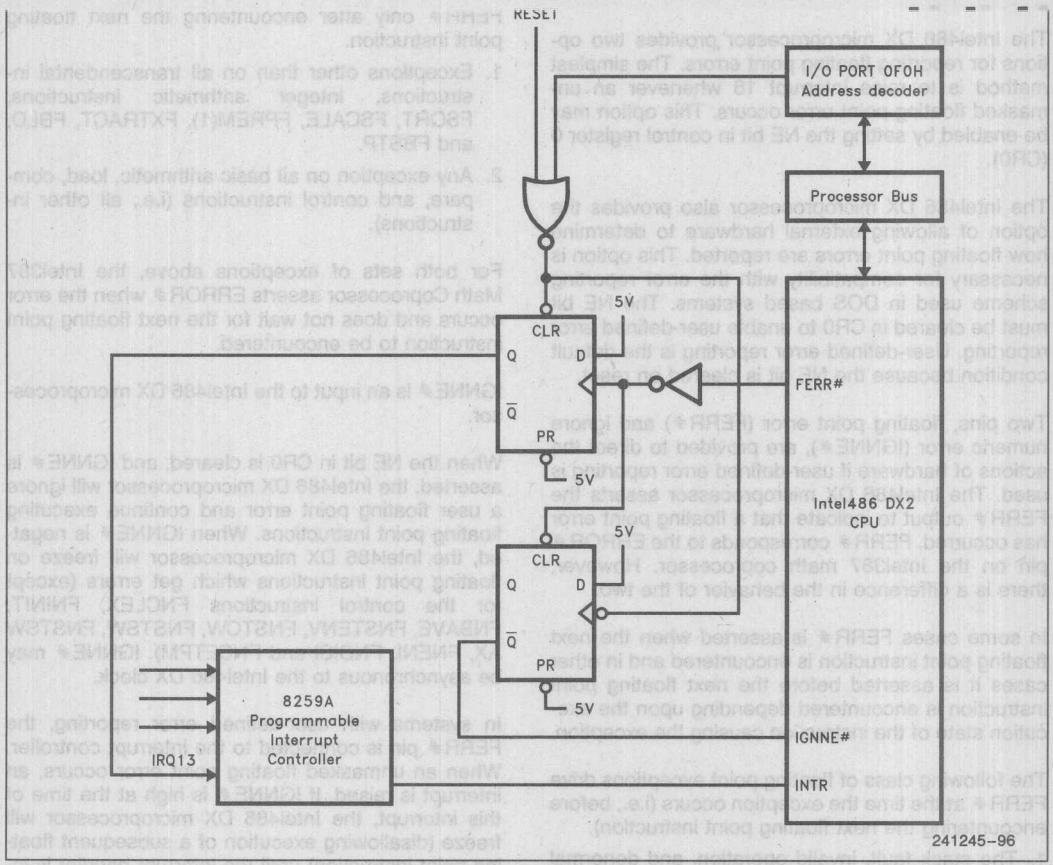


Figure 7.31. DOS Compatible Numerics Error Circuit

8.0 Intel486 DX2 CPU TESTABILITY

Testing the Intel486 DX2 microprocessor can be divided into three categories: Built-In Self Test (BIST), Boundary Scan, and external testing. BIST performs basic device testing on the Intel486 DX2 CPU, including the non-random logic, control ROM (CROM), translation lookaside buffer (TLB), and on-chip cache memory. Boundary Scan provides additional test hooks that conform to the IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std.1149.1). The Intel486 DX2 microprocessor also has a test mode in which all of its outputs are 3-stated. Additional testing can be performed by using the test registers within the Intel486 DX2 CPU.

8.1 Built-In Self Test (BIST)

The BIST is initiated by holding the AHOLD (address hold) pin HIGH for 2 CLKs before and 2 CLKs after RESET going from HIGH to LOW as shown in Figure 6.4. The BIST takes approximately 600 thousand clocks, or approximately 24 milliseconds with a 50 MHz Intel486 DX2 microprocessor. No bus cycles will be run by the Intel486 DX2 microprocessor until the BIST is concluded. Note that for the Intel486 DX2 microprocessor the RESET must be active for 15 clocks with or without BIST being enabled for warm resets.

The results of BIST is stored in the EAX register. The Intel486 DX2 microprocessor has successfully passed the BIST if the contents of the EAX register are zero. If the results in EAX are not zero then the BIST has detected a flaw in the microprocessor. The microprocessor performs reset and begins normal operation at the completion of the BIST.

The non-random logic, control ROM, on-chip cache and translation lookaside buffer (TLB) are tested during the BIST.

The cache portion of the BIST verifies that the cache is functional and that it is possible to read and write to the cache. The BIST manipulates test registers TR3, TR4 and TR5 while testing the cache. These test registers are described in Section 8.2.

The cache testing algorithm writes a value to each cache entry, reads the value back, and checks that the correct value was read back. The algorithm may be repeated more than once for each of the 512 cache entries using different constants.

The TLB portion of the BIST verifies that the TLB is functional and that it is possible to read and write to the TLB. The BIST manipulates test registers TR6 and TR7 while testing the TLB. TR6 and TR7 are described in Section 8.3.

8.2 On-Chip Cache Testing

The on-chip cache testability hooks are designed to be accessible during the BIST and for assembly language testing of the cache.

The Intel486 DX2 microprocessor contains a cache fill buffer and a cache read buffer. For testability writes, data must be written to the cache fill buffer before it can be written to a location in the cache. Data must be read from a cache location into the cache read buffer before the microprocessor can access the data. The cache fill and cache read buffer are both 128 bits wide.

8.2.1 CACHE TESTING REGISTERS TR3, TR4 AND TR5

Figure 8.1 shows the three cache testing registers: the Cache Data Test Register (TR3), the Cache Status Test Register (TR4) and the Cache Control Test Register (TR5). External access to these registers is provided through MOV reg,TREG and MOV TREG, reg instructions.

Cache Data Test Register: TR3

The cache fill buffer and the cache read buffer can only be accessed through TR3. Data to be written to the cache fill buffer must first be written to TR3. Data read from the cache read buffer must be loaded into TR3.

TR3 is 32 bits wide while the cache fill and read buffers are 128 bits wide. 32 bits of data must be written to TR3 four times to fill the cache fill buffer. 32 bits of data must be read from TR3 four times to empty the cache read buffer. The entry select bits in TR5 determine which 32 bits of data TR3 will access in the buffers.

Cache Status Test Register: TR4

TR4 handles tag, LRU and valid bit information during cache tests. TR4 must be loaded with a tag and a valid bit before a write to the cache. After a read from a cache entry, TR4 contains the tag and valid bit from that entry, and the LRU bits and four valid bits from the accessed set.

Cache Control Test Register: TR5

TR5 specifies which testability operation will be performed and the set and entry within the set which will be accessed.

The seven bit set select field determines which of the 128 sets will be accessed.

The functionality of the two entry select bits depend on the state of the control bits. When the fill or read buffers are being accessed, the entry select bits point to the 32-bit location in the buffer being accessed. When a cache location is specified, the entry select bits point to one of the four entries in a set. Refer to Table 8.1.

Five testability functions can be performed on the cache. The two control bits in TR5 specify the operation to be executed. The five operations are:

1. Write cache fill buffer
2. Perform a cache testability write
3. Perform a cache testability read
4. Read the cache read buffer
5. Perform a cache flush

Table 8.1 shows the encoding of the two control bits in TR5 for the cache testability functions. Table 8.1 also shows the functionality of the entry and set select bits for each control operation.

The cache tests attempt to use as much of the normal operating circuitry as possible. Therefore when cache tests are being performed, the cache must be disabled (the CD and NW bits in control register must be set to 1 to disable the cache. See Section 5).

8.2.2 CACHE TESTABILITY WRITE

A testability write to the cache is a two step process. First the cache fill buffer must be loaded with 128 bits of data and TR4 loaded with the tag and valid bit. Next the contents of the fill buffer are written to a cache location. Sample assembly code to do a write is given in Figure 8.2.

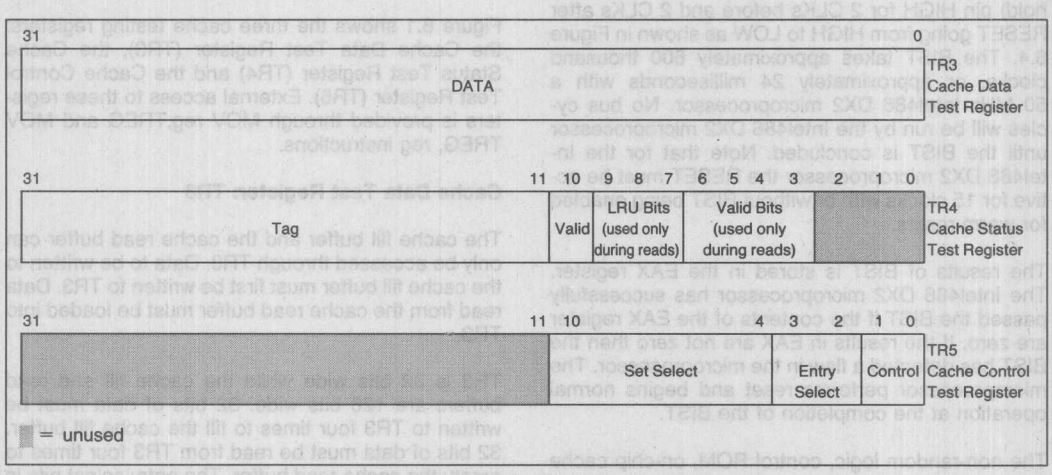


Figure 8.1. Cache Test Registers

Table 8.1. Cache Control Bit Encoding and Effect of Control Bits on Entry Select and Set Select Functionality

Control Bits		Operation	Entry Select Bits Function	Set Select Bits
Bit 1	Bit 0			
0	0	Enable { Fill Buffer Write / Read Buffer Read	Select 32-bit location in fill/read buffer	—
0	1	Perform Cache Write	Select an entry in set.	Select a set to write to
1	0	Perform Cache Read	Select an entry in set.	Select a set to read from
1	1	Perform Flush Cache	—	—

Sample Assembly Code

An example assembly language sequence to perform a cache write is:

```
;
; eax. ebx. ecx. edx contain the cache line to write
; edi contains the tag information to load
; CR0 already says to enable reads/write to TR5
;
; fill the cache buffer
        mov esi,0       ; set up command
        mov tr5,esi     ; load to TR5
        mov tr3,eax     ; load data into cache fill buffer
        mov esi,4
        mov tr5,esi
        mov tr3,ebx
        mov esi,8
        mov tr5,esi
        mov tr3,ecx
        mov esi,0ch
        mov tr5,esi
        mov tr3,edx
;
; load the Cache Status Register
;
        mov tr4,edi     ; load 21-bit tag and valid bit
;
; perform the cache write
;
        mov esi,1
        mov tr5,esi     ; write the cache (set 0, entry 0)
```

An example assembly language sequence to perform a cache read is:

```
;
; data into eax, ebx, ecx, edx; status into edi
;
; read the cache line back
;
        mov esi,2
        mov tr5,esi     ; do cache testability read (set 0, entry 0)
;
; read the data from the read buffer
;
        mov esi,0
        mov tr5,esi
        mov eax,tr3
        mov esi,4
        mov tr5,esi
        mov ebx,tr3
        mov esi,8
        mov tr5,esi
        mov ecx,tr3
        mov esi,0ch
        mov tr5,esi
        mov edx,tr3
;
; read the status from TR4
;
        mov edi,tr4
```

Figure 8.2. Sample Assembly Code for Cache Testing

Loading the fill buffer is accomplished by first writing to the entry select bits in TR5 and setting the control bits in TR5 to 00. The entry select bits identify one of four 32-bit locations in the cache fill buffer to put 32 bits of data. Following the write to TR5, TR3 is written with 32 bits of data which are immediately placed in the cache fill buffer. Writing to TR3 initiates the write to the cache fill buffer. The cache fill buffer is loaded with 128 bits of data by writing to TR5 and TR3 four times using a different entry select location each time.

TR4 must be loaded with the 21-bit tag and valid bit (bit 10 in TR4) before the contents of the fill buffer are written to a cache location.

The contents of the cache fill buffer are written to a cache location by writing TR5 with a control field of 01 along with the set select and entry select fields. The set select and entry select field indicate the location in the cache to be written. The normal cache LRU update circuitry updates the internal LRU bits for the selected set.

Note that a cache testability write can only be done when the cache is disabled for replaces (the CD bit is control register 0 is reset to 1). Also note that care must be taken when directly writing to entries in the cache. If the entry is set to overlap an area of memory that is being used in external memory, that cache entry could inadvertently be used instead of the external memory. Of course, this is exactly the type of operation that one would desire if the cache were to be used as a high speed RAM.

8.2.3 CACHE TESTABILITY READ

A cache testability read is a two step process. First the contents of the cache location are read into the cache read buffer. Next the data is examined by reading it out of the read buffer. Sample assembly code to do a testability read is given in Figure 8.2.

Reading the contents of a cache location into the cache read buffer is initiated by writing TR5 with the control bits set to 10 and the desired seven-bit set select and two-bit entry select. In response to the write to TR5, TR4 is loaded with the 21-bit tag field and the single valid bit from the cache entry read. TR4 is also loaded with the three LRU bits and four valid bits corresponding to the cache set that was accessed. The cache read buffer is filled with the 128-bit value which was found in the data array at the specified location.

The contents of the read buffer are examined by performing four reads of TR3. Before reading TR3 the entry select bits in TR5 must loaded to indicate which of the four 32-bit words in the read buffer to transfer into TR3 and the control bits in TR5 must be loaded with 00. The register read of TR3 will initiate the transfer of the 32-bit value from the read buffer to the specified general purpose register.

Note that it is very important that the entire 128-bit quantity from the read buffer and also the information from TR4 be read before any memory references are allowed to occur. If memory operations are allowed to happen, the contents of the read buffer will be corrupted. This is because the testability operations use hardware that is used in normal memory accesses for the Intel486 DX2 microprocessor whether the cache is enabled or not.

8.2.4 FLUSH CACHE

The control bits in TR5 must be written with 11 to flush the cache. None of the other bits in TR5 have any meaning when 11 is written to the control bits. Flushing the cache will reset the LRU bits and the valid bits to 0, but will not change the cache tag or data arrays.

When the cache is flushed by writing to TR5 the special bus cycle indicating a cache flush to the external system is not run (see Section 7.2.11, Special Bus Cycles). The cache should be flushed with the instruction INVD (Invalidate Data Cache) instruction or the WBINVD (Write-back and Invalidate Data Cache) instruction.

8.3 Translation Lookaside Buffer (TLB) Testing

The Intel486 DX2 microprocessor TLB testability hooks are similar to those in the Intel386 microprocessor. The testability hooks have been enhanced to provide added test features and to include new features in the Intel486 DX2 microprocessor. The TLB testability hooks are designed to be accessible during the BIST and for assembly language testing of the TLB.

8.3.1 TRANSLATION LOOKASIDE BUFFER ORGANIZATION

The Intel486 DX2 microprocessors TLB is 4-way set associative and has space for 32 entries. The TLB is logically split into three blocks shown in Figure 8.3.

The data block is physically split into four arrays, each with space for eight entries. An entry in the data block is 22 bits wide containing a 20-bit physical address and two bits for the page attributes. The page attributes are the PCD (page cache disable) bit and the PWT (page write-through) bit. Refer to Section 4.5.4 for a discussion of the PCD and PWT bits.

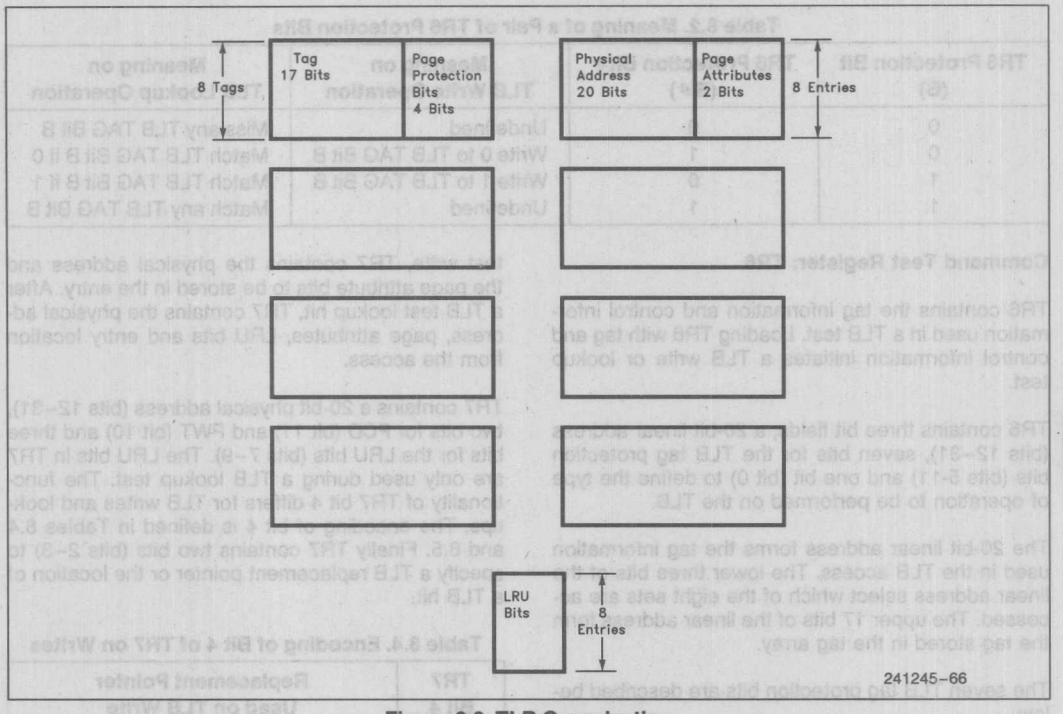


Figure 8.3. TLB Organization

The tag block is also split into four arrays, one for each of the data arrays. A tag entry is 21 bits wide containing a 17-bit linear address and four protection bits. The protection bits are valid (V), user/supervisor (U/S), read/write (R/W) and dirty (D).

The third block contains eight three bit quantities used in the pseudo least recently used (LRU) replacement algorithm. These bits are called the LRU bits. The LRU replacement algorithm used in the TLB is the same as used by the on-chip cache. For a description of this algorithm refer to Section 5.5.

8.3.2 TLB TEST REGISTERS TR6 AND TR7

The two TLB test registers are shown in Figure 8.4. TR6 is the command test register and TR7 is the data test register. External access to these registers is provided through MOV reg,TREG and MOV TREG,reg instructions.

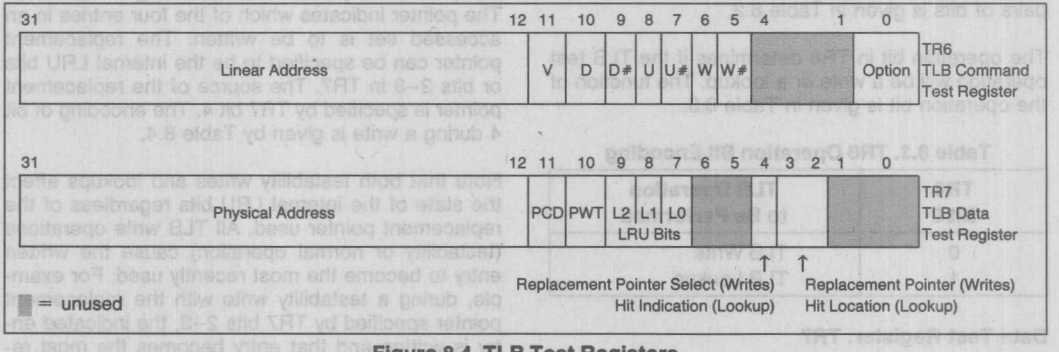


Figure 8.4. TLB Test Registers

Table 8.2. Meaning of a Pair of TR6 Protection Bits

TR6 Protection Bit (B)	TR6 Protection Bit# (B#)	Meaning on TLB Write Operation	Meaning on TLB Lookup Operation
0	0	Undefined	Miss any TLB TAG Bit B
0	1	Write 0 to TLB TAG Bit B	Match TLB TAG Bit B if 0
1	0	Write 1 to TLB TAG Bit B	Match TLB TAG Bit B if 1
1	1	Undefined	Match any TLB TAG Bit B

Command Test Register: TR6

TR6 contains the tag information and control information used in a TLB test. Loading TR6 with tag and control information initiates a TLB write or lookup test.

TR6 contains three bit fields, a 20-bit linear address (bits 12–31), seven bits for the TLB tag protection bits (bits 5-11) and one bit (bit 0) to define the type of operation to be performed on the TLB.

The 20-bit linear address forms the tag information used in the TLB access. The lower three bits of the linear address select which of the eight sets are accessed. The upper 17 bits of the linear address form the tag stored in the tag array.

The seven TLB tag protection bits are described below.

V: The valid bit for this TLB entry

D,D#: The dirty bit for/from the TLB entry

U,U#: The user/supervisor bit for/from the TLB entry

W,W#: The read/write bit for/from the TLB entry

Two bits are used to represent the D, U/S and R/W bits in the TLB tag to permit the option of a forced miss or hit during a TLB lookup operation. The forced miss or hit will occur regardless of the state of the actual bit in the TLB. The meaning of these pairs of bits is given in Table 8.2.

The operation bit in TR6 determines if the TLB test operation will be a write or a lookup. The function of the operation bit is given in Table 8.3.

Table 8.3. TR6 Operation Bit Encoding

TR6 Bit 0	TLB Operation to Be Performed
0	TLB Write
1	TLB Lookup

Data Test Register: TR7

TR7 contains the information stored or read from the data block during a TLB test operation. Before a TLB test write, TR7 contains the physical address and the page attribute bits to be stored in the entry. After a TLB test lookup hit, TR7 contains the physical address, page attributes, LRU bits and entry location from the access.

TR7 contains a 20-bit physical address (bits 12–31), two bits for PCD (bit 11) and PWT (bit 10) and three bits for the LRU bits (bits 7–9). The LRU bits in TR7 are only used during a TLB lookup test. The functionality of TR7 bit 4 differs for TLB writes and lookups. The encoding of bit 4 is defined in Tables 8.4 and 8.5. Finally TR7 contains two bits (bits 2–3) to specify a TLB replacement pointer or the location of a TLB hit.

Table 8.4. Encoding of Bit 4 of TR7 on Writes

TR7 Bit 4	Replacement Pointer Used on TLB Write
0	Pseudo-LRU Replacement Pointer
1	Data Test Register Bits 3:2

Table 8.5. Encoding of Bit 4 of TR7 on Lookups

TR7 Bit 4	Meaning after TLB Lookup Operation
0	TLB Lookup Resulted in a Miss
1	TLB Lookup Resulted in a Hit

A replacement pointer is used during a TLB write. The pointer indicates which of the four entries in an accessed set is to be written. The replacement pointer can be specified to be the internal LRU bits or bits 2–3 in TR7. The source of the replacement pointer is specified by TR7 bit 4. The encoding of bit 4 during a write is given by Table 8.4.

Note that both testability writes and lookups affect the state of the internal LRU bits regardless of the replacement pointer used. All TLB write operations (testability or normal operation) cause the written entry to become the most recently used. For example, during a testability write with the replacement pointer specified by TR7 bits 2–3, the indicated entry is written and that entry becomes the most recently used as specified by the internal LRU bits.

There are two TLB testing operations: write entries into the TLB, and perform TLB lookups. One major enhancement over TLB testing in the Intel386 microprocessor is that paging need not be disabled while executing testability writes or lookups.

Note that any time one TLB set contains the same linear address in more than one of its entries, looking up that linear address will give unpredictable results. Therefore a single linear address should not be written to one TLB set more than once.

8.3.3 TLB WRITE TEST

To perform a TLB write TR7 must be loaded followed by a TR6 load. The register operations must be performed in this order since the TLB operation is triggered by the write to TR6.

TR7 is loaded with a 20-bit physical address and values for PCD and PWT to be written to the data portion of the TLB. In addition, bit 4 of TR7 must be loaded to indicate whether to use TR7 bits 3-2 or the internal LRU bits as the replacement pointer on the TLB write operation. Note that the LRU bits in TR7 are not used in a write test.

TR6 must be written to initiate the TLB write operation. Bit 0 in TR6 must be reset to zero to indicate a TLB write. The 20-bit linear address and the seven page protection bits must also be written in TR6 to specify the tag portion of the TLB entry. Note that the three least significant bits of the linear address specify which of the eight sets in the data block will be loaded with the physical address data. Thus only 17 of the linear address bits are stored in the tag array.

8.3.4 TLB LOOKUP TEST

To perform a TLB lookup it is only necessary to write the proper tags and control information into TR6. Bit 0 in TR6 must be set to 1 to indicate a TLB lookup. TR6 must be loaded with a 20-bit linear address and the seven protection bits. To force misses and matches of the individual protection bits on TLB lookups, set the seven protection bits as specified in Table 8.2.

A TLB lookup operation is initiated by the write to TR6. TR7 will indicate the result of the lookup operation following the write to TR6. The hit/miss indication can be found in TR7 bit 4 (see Table 8.5).

TR7 will contain the following information if bit 4 indicated that the lookup test resulted in a hit. Bits 2–3 will indicate in which set the match occurred. The 22 most significant bits in TR7 will contain the physical address and page attributes contained in the entry. Bits 9–7 will contain the LRU bits associated with the accessed set. The state of the LRU bits is previous to their being updated for the current lookup.

If bit 4 in TR7 indicated that the lookup test resulted in a miss the remaining bits in TR7 are undefined.

Again it should be noted that a TLB testability lookup operation affects the state of the LRU bits. The LRU bits will be updated if a hit occurred. The entry which was hit will become the most recently used.

8.4 3-State Output Test Mode

The Intel486 DX2 microprocessor provides the ability to float all its outputs and bidirectional pins. This includes all pins floated during bus hold as well as pins which are never floated in normal operation of the chip (HLDA, BREQ, FERR# and PCHK#). When the Intel486 DX2 microprocessor is in the 3-state output test mode external testing can be used to test board connections.

The 3-state test mode is invoked by driving FLUSH# low for 2 clocks before and 2 clocks after RESET going low. The outputs are guaranteed to 3-state no later than 10 clocks after RESET goes low (see Figure 6.4). The Intel486 DX2 microprocessor remains in the 3-state test mode until the next RESET.

8.5 Intel486™ DX2 Microprocessor Boundary Scan (JTAG)

The Intel486 DX2 microprocessor provides testability features compatible with the IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std.1149.1). The test logic provided allows for testing to insure that components function correctly, that interconnections between various components are correct, and that various components interact correctly on the printed circuit board.

The boundary scan test logic consists of a boundary scan register and support logic that are accessed through a test access port (TAP). The TAP provides a simple serial interface that makes it possible to test all signal traces with only a few probes.

The TAP can be controlled via a bus master. The bus master can be either automatic test equipment or a component (PLD) that interfaces to the four-pin test bus.

8.5.1 BOUNDARY SCAN ARCHITECTURE

The boundary scan test logic contains the following elements:

— Test access port (TAP), consisting of input pins TMS, TCK, and TDI; and output pin TDO.

— TAP controller, which interprets the inputs on the test mode select (TMS) line and performs the corresponding operation. The operations performed by the TAP include controlling the instruction and data registers within the component.

— Instruction register (IR), which accepts instruction codes shifted into the test logic on the test data input (TDI) pin. The instruction codes are used to select the specific test operation to be performed or the test data register to be accessed.

— Test data registers: The Intel486 DX2 microprocessor contains three test data registers: Bypass register (BPR), Device Identification register (DID), and Boundary Scan register (BSR).

The instruction and test data registers are separate shift-register paths connected in parallel and have a common serial data input and a common serial data output connected to the TAP signals, TDI and TDO, respectively.

8.5.2 DATA REGISTERS

The Intel486 DX2 CPU contains the two required test data registers; bypass register and boundary scan register. In addition, they also have a device identification register.

Each test data register is serially connected to TDI and TDO, with TDI connected to the most significant bit and TDO connected to the least significant bit of the test data register. Data is shifted one stage (bit position within the register) on each rising edge of the test clock (TCK).

In addition the Intel486 DX2 CPU contains a runbist register to support the RUNBIST boundary scan instruction.

8.5.2.1 Bypass Register

The Bypass Register is a one-bit shift register that provides the minimal length path between TDI and TDO. This path can be selected when no test operation is being performed by the component to allow rapid movement of test data to and from other components on the board. While the bypass register is selected, data is transferred from TDI to TDO without inversion.

8.5.2.2 Boundary Scan Register

The Boundary Scan Register is a single shift register path containing the boundary scan cells that are connected to all input and output pins of the Intel486 DX2 CPU. Figure 8.5 shows the logical structure of the boundary scan register. While output cells determine the value of the signal driven on the corresponding pin, input cells only capture data; they do not affect the normal operation of the device. Data is transferred without inversion from TDI to TDO through the boundary scan register during scanning. The boundary scan register can be operated by the EXTEST and SAMPLE instructions. The boundary scan register order is described in Section 8.5.5.

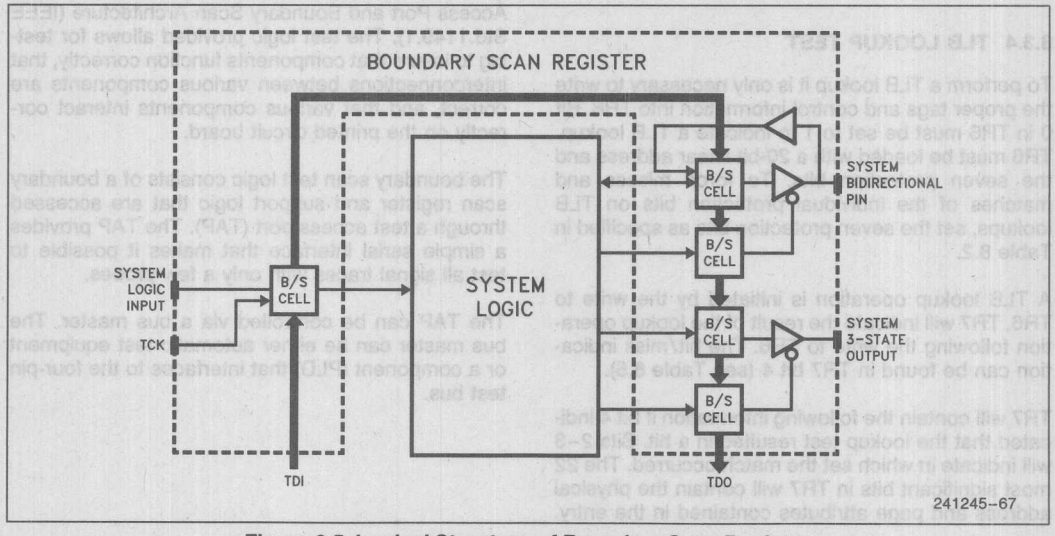


Figure 8.5. Logical Structure of Boundary Scan Register

8.5.2.3 Device Identification Register

The Device Identification Register contains the manufacturer's identification code, part number code, and version code in the format shown in Figure 8.6. Table 8.6 lists the codes corresponding to the Intel486 DX2 CPU.

8.5.2.4 Runbist Register

The Runbist Register is a one bit register used to report the results of the Intel486 DX2 CPU BIST when it is initiated by the RUNBIST instruction. This register is loaded with a "1" prior to invoking the BIST and is loaded with "0" upon successful completion.

8.5.3 INSTRUCTION REGISTER

The Instruction Register (IR) allows instructions to be serially shifted into the device. The instruction selects the particular test to be performed, the test data register to be accessed, or both. The instruction register is four (4) bits wide. The most significant bit is connected to TDI and the least significant bit is connected to TDO. There are no parity bits associated with the Instruction register. Upon entering the Capture-IR TAP controller state, the Instruction register is loaded with the default instruction "0001", SAMPLE/PRELOAD. Instructions are shifted into the instruction register on the rising edge of TCK while the TAP controller is in the Shift-IR state.

8.5.3.1 Intel486 DX2 CPU Boundary Scan Instruction Set

The Intel486 DX2 CPU supports all three mandatory boundary scan instructions (BYPASS, SAMPLE/PRELOAD, and EXTEST) along with two optional instructions (IDCODE and RUNBIST). Table 8.7 lists the Intel486 DX2 CPU boundary scan instruction codes. The instructions listed as PRIVATE cause TDO to become enabled in the Shift-DR state and cause "0" to be shifted out of TDO on the rising edge of TCK. Execution of the PRIVATE instructions will not cause hazardous operation of the Intel486 DX2 CPU.

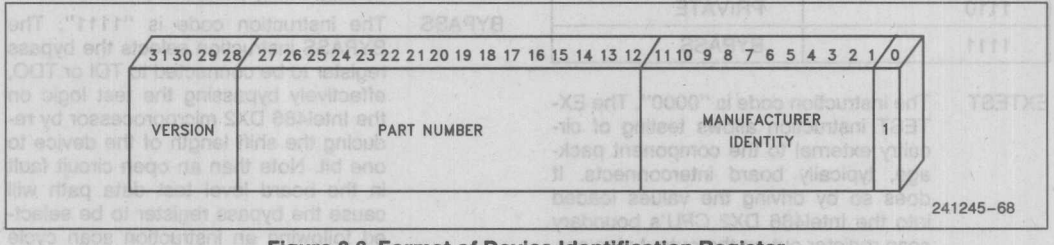


Figure 8.6. Format of Device Identification Register

Table 8.6. Device Identification Codes

Component Code	Version Code	Part Number Code	Manufacturer Identity
Intel486 DX2 CPU (Ax)	00h	0432h	09h
Intel486 DX2 CPU (Bx)	00h	0433h	09h

Table 8.7. Boundary Scan Instruction Codes

Instruction Code	Instruction Name
0000	EXTEST
0001	SAMPLE
0010	IDCODE
0011	PRIVATE
0100	PRIVATE
0101	PRIVATE
0110	PRIVATE
0111	PRIVATE
1000	RUNBIST
1001	PRIVATE
1010	PRIVATE
1011	PRIVATE
1100	PRIVATE
1101	PRIVATE
1110	PRIVATE
1111	BYPASS

EXTEST The instruction code is "0000". The EXTEST instruction allows testing of circuitry external to the component package, typically board interconnects. It does so by driving the values loaded into the Intel486 DX2 CPU's boundary scan register out on the output pins corresponding to each boundary scan cell and capturing the values on Intel486 DX2 CPU input pins to be loaded into their corresponding boundary scan register locations. I/O pins are selected as input or output, depending on the value loaded into their control setting locations in the boundary scan register. Values shifted into input latches in the boundary scan register are never used by the internal logic of the Intel486 DX2 CPU.

NOTE:

After using the EXTEST instruction, the Intel486 DX2 CPU must be reset before normal (non-boundary scan) use.

SAMPLE/PRELOAD The instruction code is "0001". The SAMPLE/PRELOAD has two functions that it performs. When the TAP controller is in the Capture-DR state, the SAMPLE/PRELOAD instruction allows a "snap-shot" of the normal operation of the component without interfering with that normal operation. The instruction causes boundary scan register cells associated with outputs to sample the value being driven by the Intel486 DX2 CPU. It causes the cells associated with inputs to sample the value being driven into the Intel486 DX2 CPU. On both outputs and inputs the sampling occurs on the rising edge of TCK. When the TAP controller is in the Update-DR state, the SAMPLE/PRELOAD instruction preloads data to the device pins to be driven to the board by executing the EXTEST instruction. Data is preloaded to the pins from the boundary scan register on the falling edge of TCK.

IDCODE The instruction code is "0010". The IDCODE instruction selects the device identification register to be connected to TDI and TDO, allowing the device identification code to be shifted out of the device on TDO. Note that the device identification register is not altered by data being shifted in on TDI.

BYPASS The instruction code is "1111". The BYPASS instruction selects the bypass register to be connected to TDI or TDO, effectively bypassing the test logic on the Intel486 DX2 microprocessor by reducing the shift length of the device to one bit. Note than an open circuit fault in the board level test data path will cause the bypass register to be selected following an instruction scan cycle due to the pull-up resistor on the TDI input. This has been done to prevent any unwanted interference with the proper operation of the system logic.

RUNBIST The instruction code is "1000". The RUNBIST instruction selects the one (1) bit runbist register, loads a value of "1" into the runbist register, and connects it to TDO. It also initiates the built-in self test (BIST) feature of the Intel486 DX2 CPU, which is able to detect approximately 60% of the stuck-at faults on the Intel486 DX2 CPU. The Intel486 DX2 CPU AC/DC Specifications for V_{CC} and CLK must be met and reset must have been asserted at least once prior to executing the RUNBIST boundary scan instruction. After loading the RUNBIST instruction code in the instruction register, the TAP controller must be placed in the Run-Test/Idle state. BIST begins on the first rising edge of TCK after entering the Run-Test/Idle state. The TAP

controller must remain in the Run-Test/Idle state until BIST is completed. It requires 1.2 million clock (CLK) cycles to complete BIST and report the result to the runbist register. After completing the 1.2 million clock (CLK) cycles, the value in the runbist register should be shifted out on TDO during the Shift-DR state. A value of "0" being shifted out on TDO indicates BIST successfully completed. A value of "1" indicates a failure occurred. After executing the RUNBIST instruction, the Intel486 DX2 CPU must be reset prior to normal operation.

8.5.4 TEST ACCESS PORT (TAP) CONTROLLER

The TAP controller is a synchronous, finite state machine. It controls the sequence of operations of the test logic. The TAP controller changes state only in response to the following events:

1. a rising edge of TCK
2. power-up.

The value of the test mode state (TMS) input signal at a rising edge of TCK controls the sequence of the state changes. The state diagram for the TAP controller is shown in Figure 8.7. Test designers must consider the operation of the state machine in order to design the correct sequence of values to drive on TMS.

8.5.4.1 Test-Logic-Reset State

In this state, the test logic is disabled so that normal operation of the device can continue unhindered. This is achieved by initializing the instruction register such that the IDCODE instruction is loaded. No matter what the original state of the controller, the controller enters Test-Logic-Reset state when the TMS input is held high (1) for at least five rising edges of TCK. The controller remains in this state while TMS is high. The TAP controller is also forced to enter this state at power-up.

8.5.4.2 Run-Test/Idle State

A controller state between scan operations. Once in this state, the controller remains in this state as long

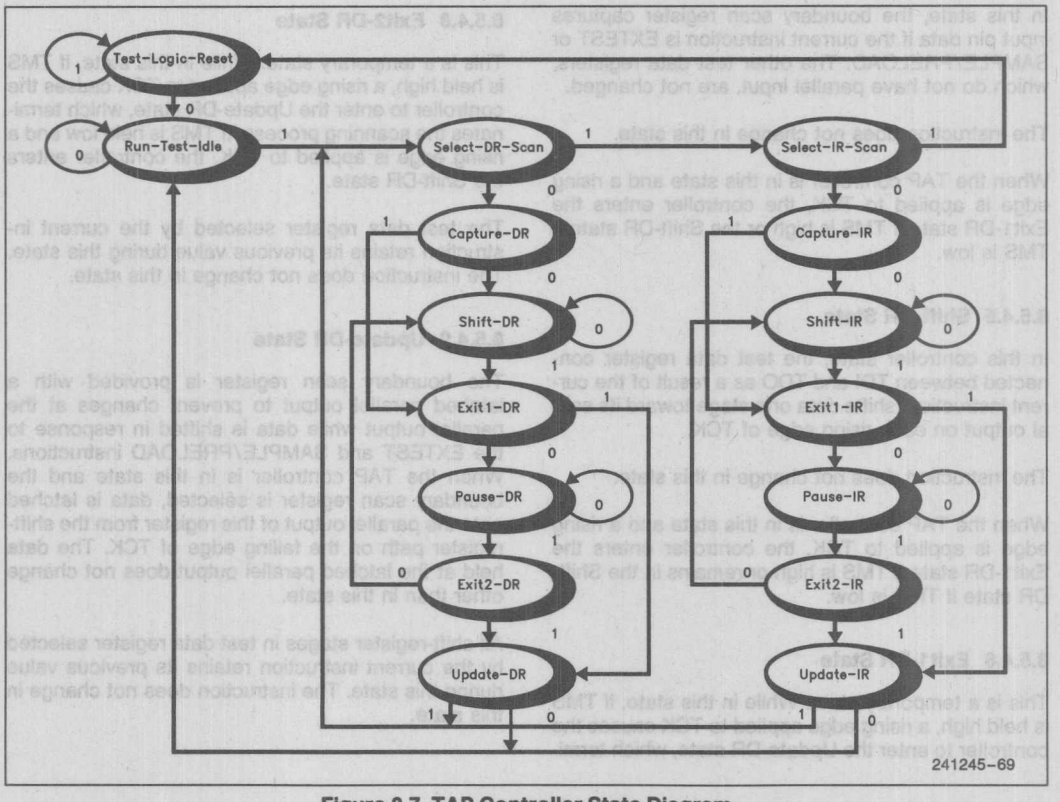


Figure 8.7. TAP Controller State Diagram

as TMS is held low. In devices supporting the RUNBIST instruction, the BIST is performed during this state and the result is reported in the runbist register. For instruction not causing functions to execute during this state, no activity occurs in the test logic. The instruction register and all test data registers retain their previous state. When TMS is high and a rising edge is applied to TCK, the controller moves to the Select-DR state.

8.5.4.3 Select-DR-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held low and a rising edge is applied to TCK when in this state, the controller moves into the Capture-DR state, and a scan sequence for the selected test data register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves to the Select-IR-Scan state.

The instruction does not change in this state.

8.5.4.4 Capture-DR State

In this state, the boundary scan register captures input pin data if the current instruction is EXTEST or SAMPLE/PRELOAD. The other test data registers, which do not have parallel input, are not changed.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-DR state if TMS is high or the Shift-DR state if TMS is low.

8.5.4.5 Shift-DR State

In this controller state, the test data register connected between TDI and TDO as a result of the current instruction, shifts data one stage toward its serial output on each rising edge of TCK.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-DR state if TMS is high or remains in the Shift-DR state if TMS is low.

8.5.4.6 Exit1-DR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-DR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.7 Pause-Dr State

The pause state allows the test controller to temporarily halt the shifting of data through the test data register in the serial path between TDI and TDO. An example of using this state could be to allow a tester to reload its pin memory from disk during application of a long test sequence.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves to the Exit2-DR state.

8.5.4.8 Exit2-DR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Shift-DR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.9 Update-DR State

The boundary scan register is provided with a latched parallel output to prevent changes at the parallel output while data is shifted in response to the EXTEST and SAMPLE/PRELOAD instructions. When the TAP controller is in this state and the boundary scan register is selected, data is latched onto the parallel output of this register from the shift-register path on the falling edge of TCK. The data held at the latched parallel output does not change other than in this state.

All shift-register stages in test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.10 Select-IR-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held low and a rising edge is applied to TCK when in this state, the controller moves into the Capture-IR state, and a scan sequence for the instruction register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves to the Test-Logic-Reset state.

The instruction does not change in this state.

8.5.4.11 Capture-IR State

In this controller state the shift register contained in the instruction register loads the fixed value "0001" on the rising edge of TCK.

The test data register selected by the current instruction retains it previous value during this state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-IR state if TMS is held high, or the Shift-IR state if TMS is held low.

8.5.4.12 Shift-IR State

In this state the shift register contained in the instruction register is connected between TDI and TDO and shifts data one stage towards its serial output on each rising edge of TCK.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-IR state if TMS is held high, or remains in the Shift-IR state if TMS is held low.

8.5.4.13 Exit1-IR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-IR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.14 Pause-IR State

The pause state allows the test controller to temporarily halt the shifting of data through the instruction register.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves to the Exit2-IR state.

8.5.4.15 Exit2-IR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Shift-IR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.16 Update-IR State

The instruction shifted into the instruction register is latched onto the parallel output from the shift-register path on the falling edge of TCK. Once the new instruction has been latched, it becomes the current instruction.

Test data registers selected by the current instruction retain the previous value.

8.5.5 BOUNDARY SCAN REGISTER CELL

The boundary scan register contains a cell for each pin, as well as cells for control of I/O and 3-state pins.

The following is the bit order of the Intel486 DX2 CPU boundary scan register: (from left to right and top to bottom).

TDI → WRCTL ABUSCTL BUSCTL MISCCTL ADS# BLAST# PLOCK# LOCK# PCHK# BRDY# BOFF# BS16# BS8# RDY# KEN# HOLD AHOLD CLK HLDA WR# BREQ BE0# BE1# BE2# BE3# MIO# DC# PWT PCD EADS# A20M# RESET FLUSH# INTR NMI UP# FERR# IGNNE# D31 D30 D29 D28 D27 D26 D25 D24 DP3 D23 D22 D21 D20 D19 D18 D17 D16 DP2 D15 D14 D13 D12 D11 D10 D9 D8 DP1 D7 D6 D5 D4 D3 D2 D1 D0 DP0 A31 A30 A29 A28 A27 A26 A25 A24 A23 A22 A21 A20 A19 A18 A17 A16 A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 RESERVED A5 A4 A3 A2 → TDO

"RESERVED" corresponds to no connect "NC" signals on the Intel486 DX2 CPU.

All the *CTL cells are control cells that are used to select the direction of bidirectional pins or 3-state output pins. If "1" is loaded into the control cell (*CTL), the associated pin(s) are 3-stated or selected as input. The following lists the control cells and their corresponding pins.

1. WRCTL controls the D31–0 and DP3–0 pins.
2. ABUSCTL controls the A31–A2 pins.
3. BUSCTL controls the ADS#, BLAST#, PLOCK#, LOCK#, WR#, BE0#, BE1#, BE2#, BE3#, MIO#, DC#, PWT, and PCD pins.
4. MISCCTL controls the PCHK#, HLDA, BREQ, and FERR# pins.

8.5.6 TAP CONTROLLER INITIALIZATION

The TAP controller is automatically initialized when a device is powered up. In addition, the TAP controller can be initialized by applying a high signal level on the TMS input for five TCK periods.

8.5.7 BOUNDARY SCAN DESCRIPTION LANGUAGE (BSDL)

```
-- Intel i486(tm)DX2 CPU BSDL description

--

entity Intel486TM_DX2 is
   generic(PHYSICAL_PIN_MAP : string := "PGA_17x17");

   port (A20M      : in    bit;
         ABUS2     : out   bit;
         ABUS3     : out   bit;
         ABUS      : inout bit_vector (4 to 31); -- Address bus (words)
         ADS       : out   bit;
         AHOLD     : in    bit;
         BE        : out   bit_vector(0 to 3);
         BLAST     : out   bit;
         BOFF      : in    bit;
         BRDY      : in    bit;
         BREQ      : out   bit;
         BS8       : in    bit;
         BS16      : in    bit;
         CLK       : in    bit;
         DBUS      : inout bit_vector(0 to 31);  -- Data bus
         DC        : out   bit;
         DP        : inout bit_vector(0 to 3);
         EADS      : in    bit;
         FERR      : out   bit;
         FLUSH     : in    bit;
         HLDA      : out   bit;
         HOLD      : in    bit;
         IGNNE     : in    bit;
         INTR      : in    bit;
         KEN       : in    bit;
         LOCK      : out   bit;
         MIO       : out   bit;
         NC        : linkage bit_vector(1 to 12); -- No Connects
         NC1       : in    bit;
         NMI       : in    bit;
         PCD       : out   bit;
         PCHK      : out   bit;
         PLOCK     : out   bit;
         PWT       : out   bit;
         RDY       : in    bit;
         RESET     : in    bit;
         TCK, TMS, TDI : in bit;          -- Scan Port inputs
         TDO       : out   bit;           -- Scan Port output
         UP        : in    bit;
         VCC       : linkage bit_vector(1 to 24);   -- VCC
         VSS       : linkage bit_vector(1 to 28);   -- VSS
         WR        : out   bit);

   use STD_1149_1_1990.all;

   attribute PIN_MAP of Intel486TM_DX2 : entity is PHYSICAL_PIN_MAP;

   constant PGA_17x17 : PIN_MAP_STRING :=        -- Define Pin Out of PGA
        "A20M      : D15, "  &
        "ABUS2     : Q14, "  &
        "ABUS3     : R15, "  &
        "ABUS      : (S16, Q12, S15, Q13, R13, Q11, S13, R12," &
        "             S7,  Q10, S5,  R7,  Q9,  Q3,  R5,  Q4,  Q8,  Q5," &
        "             Q7,  S3,  Q6,  R2,  S2,  S1,  R1,  P2,  P3,  Q1)," &
        "ADS       : S17, "  &
        "AHOLD     : A17, "  &
        "BE        : (K15, J16, J15, F17), "  &
```

241245-97

Intel486™ DX2 CPU BSDL Model for Boundary Scan

SEE PRELIMINARY INTEL FOR DESIGN-IN INFORMATION

```
        "BLAST     : R16, "  &
        "BOFF      : D17, "  &
        "BRDY      : H15, "  &
        "BREQ      : Q15, "  &
        "BS8       : D16, "  &
        "BS16      : C17, "  &
        "CLK       : C3,  "  &
        "DBUS      : (P1, N2, N1, H2, M3, J2, L2, L3, F2, D1, E3, "  &
        "            C1, G3, D2, K3, F3, J3, D3, C2, B1, A1, B2, "  &
        "            A2, A4, A6, B6, C7, C6, C8, A8, C9, B8)," &
        "DC        : M15, "  &
        "DP        : (N3,  F1,  H3,  A5), "  &
        "EADS      : B17, "  &
        "FERR      : C14, "  &
        "FLUSH     : C15, "  &
        "HLDA      : P15, "  &
        "HOLD      : E15, "  &
        "IGNNE     : A15, "  &
        "INTR      : A16, "  &
        "KEN       : F15, "  &
        "LOCK      : N15, "  &
        "MIO       : N16, "  &
        "NC        : (R17, G15, C10, C12, C13, B10, B12, B13," &
        "            A10, A12, A13)," &
        "NC1       : S4,  "  &
        "NMI       : B15, "  &
        "PCD       : J17, "  &
        "PCHK      : Q17, "  &
        "PLOCK     : Q16, "  &
        "PWT       : L15, "  &
        "RDY       : F16, "  &
        "RESET     : C16, "  &
        "TCK       : A3,  "  &
        "TDI       : A14, "  &
        "TDO       : B16, "  &
        "TMS       : B14, "  &
        "UP        : C11, "  &
        "VCC       : (R8,  R9,  R10, R11, R14, P16, M2,  M16, L16, K2,  K16, " &
        "            J1,  H16, G2,  G16, E2,  E16, C4,  C5,  B7,  B9,  B11, " &
        "            R3,  R6), " &
        "VSS       : (S6,  S8,  S9,  S10, S11, S12, S14, R4,  Q2,  P17, M1,  " &
        "            M17, L1,  L17, K1,  K17, H1,  H17, G1,  G17, E1,  E17, " &
        "            B3,  B4,  B5,  A7,  A9,  A11), "  &
        "WR        : N17 *";

attribute Tap_Scan_In of     TDI      : signal is true;
attribute Tap_Scan_Mode of   TMS      : signal is true;
attribute Tap_Scan_Out of    TDO      : signal is true;
attribute Tap_Scan_Clock of  TCK      : signal is (25.0e6, BOTH);

attribute Instruction_Length of Intel486TM_DX2: entity is 4;

attribute Instruction_Opcode of Intel486TM_DX2: entity is
   "BYPASS  (1111),"  &
   "EXTEST  (0000),"  &
   "SAMPLE  (0001),"  &
   "IDCODE  (0010),"  &
   "RUNBIST (1000),"  &
   "PRIVATE (0011,0100,0101,0110,0111,1001,1010,1011,1100,1101,1110)";

attribute Instruction_Capture of Intel486TM_DX2: entity is "0001";
-- there is no Instruction_Disable attribute for Intel486TM_DX2

attribute Instruction_Private of Intel486TM_DX2: entity is "private";

attribute Instruction_Usage of Intel486TM_DX2: entity is
   "RUNBIST (registers BIST; "  &
   "result 0;"                  &
   "clock CLK in Run_Test_Idle;"&
   "length 1200000)";

attribute Idcode_Register of Intel486TM_DX2: entity is
   "0000"                 &  --version
   "0000010000010001"     &  --new part number
   "00000001001"          &  --manufacturers identity
   "1";                      --required by the standard

attribute Register_Access of Intel486TM_DX2: entity is
   "BIST[1] (RUNBIST)" ;
```

241245-98

* PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

Intel486™ DX2 CPU BSDL Model for Boundary Scan (Continued)

```
--(*********************************************************************)
--(  The first cell is closest to TDO                                   )
--(*********************************************************************)

   attribute Boundary_Length of Intel486TM_DX2: entity is 105;
   attribute Boundary_Cells of Intel486TM_DX2: entity is "BC_2, BC_1, BC_6";

   attribute Boundary_Register of Intel486TM_DX2: entity is
      "0   (BC_2,  ABUS2,       output3, X,  102,  1,  Z),"  &
      "1   (BC_2,  ABUS3,       output3, X,  102,  1,  Z),"  &
      "2   (BC_6,  ABUS(4),     bidir,  X,  102,  1,  Z),"   &
      "3   (BC_6,  ABUS(5),     bidir,  X,  102,  1,  Z),"   &
      "4   (BC_1,  NC1,         input,  X),"   &
      "5   (BC_6,  ABUS(6),     bidir,  X,  102,  1,  Z),"   &
      "6   (BC_6,  ABUS(7),     bidir,  X,  102,  1,  Z),"   &
      "7   (BC_6,  ABUS(8),     bidir,  X,  102,  1,  Z),"   &
      "8   (BC_6,  ABUS(9),     bidir,  X,  102,  1,  Z),"   &
      "9   (BC_6,  ABUS(10),    bidir,  X,  102,  1,  Z),"   &
      "10  (BC_6,  ABUS(11),    bidir,  X,  102,  1,  Z),"   &
      "11  (BC_6,  ABUS(12),    bidir,  X,  102,  1,  Z),"   &
      "12  (BC_6,  ABUS(13),    bidir,  X,  102,  1,  Z),"   &
      "13  (BC_6,  ABUS(14),    bidir,  X,  102,  1,  Z),"   &
      "14  (BC_6,  ABUS(15),    bidir,  X,  102,  1,  Z),"   &
      "15  (BC_6,  ABUS(16),    bidir,  X,  102,  1,  Z),"   &
      "16  (BC_6,  ABUS(17),    bidir,  X,  102,  1,  Z),"   &
      "17  (BC_6,  ABUS(18),    bidir,  X,  102,  1,  Z),"   &
      "18  (BC_6,  ABUS(19),    bidir,  X,  102,  1,  Z),"   &
      "19  (BC_6,  ABUS(20),    bidir,  X,  102,  1,  Z),"   &
      "20  (BC_6,  ABUS(21),    bidir,  X,  102,  1,  Z),"   &
      "21  (BC_6,  ABUS(22),    bidir,  X,  102,  1,  Z),"   &
      "22  (BC_6,  ABUS(23),    bidir,  X,  102,  1,  Z),"   &
      "23  (BC_6,  ABUS(24),    bidir,  X,  102,  1,  Z),"   &
      "24  (BC_6,  ABUS(25),    bidir,  X,  102,  1,  Z),"   &
      "25  (BC_6,  ABUS(26),    bidir,  X,  102,  1,  Z),"   &
      "26  (BC_6,  ABUS(27),    bidir,  X,  102,  1,  Z),"   &
      "27  (BC_6,  ABUS(28),    bidir,  X,  102,  1,  Z),"   &
      "28  (BC_6,  ABUS(29),    bidir,  X,  102,  1,  Z),"   &
      "29  (BC_6,  ABUS(30),    bidir,  X,  102,  1,  Z),"   &
      "30  (BC_6,  ABUS(31),    bidir,  X,  102,  1,  Z),"   &
      "31  (BC_6,  DP(0),       bidir,  X,  103,  1,  Z),"   &
      "32  (BC_6,  DBUS(0),     bidir,  X,  103,  1,  Z),"   &
      "33  (BC_6,  DBUS(1),     bidir,  X,  103,  1,  Z),"   &
      "34  (BC_6,  DBUS(2),     bidir,  X,  103,  1,  Z),"   &
      "35  (BC_6,  DBUS(3),     bidir,  X,  103,  1,  Z),"   &
      "36  (BC_6,  DBUS(4),     bidir,  X,  103,  1,  Z),"   &
      "37  (BC_6,  DBUS(5),     bidir,  X,  103,  1,  Z),"   &
      "38  (BC_6,  DBUS(6),     bidir,  X,  103,  1,  Z),"   &
      "39  (BC_6,  DBUS(7),     bidir,  X,  103,  1,  Z),"   &
      "40  (BC_6,  DP(1),       bidir,  X,  103,  1,  Z),"   &
      "41  (BC_6,  DBUS(8),     bidir,  X,  103,  1,  Z),"   &
      "42  (BC_6,  DBUS(9),     bidir,  X,  103,  1,  Z),"   &
      "43  (BC_6,  DBUS(10),    bidir,  X,  103,  1,  Z),"   &
      "44  (BC_6,  DBUS(11),    bidir,  X,  103,  1,  Z),"   &
      "45  (BC_6,  DBUS(12),    bidir,  X,  103,  1,  Z),"   &
      "46  (BC_6,  DBUS(13),    bidir,  X,  103,  1,  Z),"   &
      "47  (BC_6,  DBUS(14),    bidir,  X,  103,  1,  Z),"   &
      "48  (BC_6,  DBUS(15),    bidir,  X,  103,  1,  Z),"   &
      "49  (BC_6,  DP(2),       bidir,  X,  103,  1,  Z),"   &
      "50  (BC_6,  DBUS(16),    bidir,  X,  103,  1,  Z),"   &
      "51  (BC_6,  DBUS(17),    bidir,  X,  103,  1,  Z),"   &
      "52  (BC_6,  DBUS(18),    bidir,  X,  103,  1,  Z),"   &
      "53  (BC_6,  DBUS(19),    bidir,  X,  103,  1,  Z),"   &
      "54  (BC_6,  DBUS(20),    bidir,  X,  103,  1,  Z),"   &
      "55  (BC_6,  DBUS(21),    bidir,  X,  103,  1,  Z),"   &
      "56  (BC_6,  DBUS(22),    bidir,  X,  103,  1,  Z),"   &
      "57  (BC_6,  DBUS(23),    bidir,  X,  103,  1,  Z),"   &
      "58  (BC_6,  DP(3),       bidir,  X,  103,  1,  Z),"   &
      "59  (BC_6,  DBUS(24),    bidir,  X,  103,  1,  Z),"   &
      "60  (BC_6,  DBUS(25),    bidir,  X,  103,  1,  Z),"   &
      "61  (BC_6,  DBUS(26),    bidir,  X,  103,  1,  Z),"   &
      "62  (BC_6,  DBUS(27),    bidir,  X,  103,  1,  Z),"   &
      "63  (BC_6,  DBUS(28),    bidir,  X,  103,  1,  Z),"   &
      "64  (BC_6,  DBUS(29),    bidir,  X,  103,  1,  Z),"   &
      "65  (BC_6,  DBUS(30),    bidir,  X,  103,  1,  Z),"   &
      "66  (BC_6,  DBUS(31),    bidir,  X,  103,  1,  Z),"   &
      "67  (BC_1,  IGNNE,       input, X),"   &
      "68  (BC_2,  FERR,        output3, X,  100,  1,  Z),"   &
      "69  (BC_1,  UP,          input, X),"   &
      "70  (BC_1,  NMI,         input, X),"   &
      "71  (BC_1,  INTR,        input, X),"   &
```

241245-99

SEE PRELIMINARY INTEL FOR DESIGN-IN INFORMATION

Intel486™ DX2 CPU BSDL Model for Boundary Scan (Continued)

```
        "72  (BC_1,   FLUSH,      input, X)," &
        "73  (BC_1,   RESET,      input, X)," &
        "74  (BC_1,   A20M,       input, X)," &
        "75  (BC_1,   EADS,       input, X)," &
        "76  (BC_2,   PCD,        output3, X,  101,  1,  Z)," &
        "77  (BC_2,   PWT,        output3, X,  101,  1,  Z)," &
        "78  (BC_2,   DC,         output3, X,  101,  1,  Z)," &
        "79  (BC_2,   MIO,        output3, X,  101,  1,  Z)," &
        "80  (BC_2,   BE(3),      output3, X,  101,  1,  Z)," &
        "81  (BC_2,   BE(2),      output3, X,  101,  1,  Z)," &
        "82  (BC_2,   BE(1),      output3, X,  101,  1,  Z)," &
        "83  (BC_2,   BE(0),      output3, X,  101,  1,  Z)," &
        "84  (BC_2,   BREQ,       output3, X,  100,  1,  Z)," &
        "85  (BC_2,   WR,         output3, X,  101,  1,  Z)," &
        "86  (BC_2,   HLDA,       output3, X,  100,  1,  Z)," &
        "87  (BC_1,   CLK,        input, X)," &
        "88  (BC_1,   AHOLD,      input, X)," &
        "89  (BC_1,   HOLD,       input, X)," &
        "90  (BC_1,   KEN,        input, X)," &
        "91  (BC_1,   RDY,        input, X)," &
        "92  (BC_1,   BS8,        input, X)," &
        "93  (BC_1,   BS16,       input, X)," &
        "94  (BC_1,   BOFF,       input, X)," &
        "95  (BC_1,   BRDY,       input, X)," &
        "96  (BC_2,   PCHK,       output3, X,  100,  1,  Z)," &
        "97  (BC_2,   LOCK,       output3, X,  101,  1,  Z)," &
        "98  (BC_2,   PLOCK,      output3, X,  101,  1,  Z)," &
        "99  (BC_2,   BLAST,      output3, X,  101,  1,  Z)," &
        "100 (BC_2,   ADS,        output3, X,  101,  1,  Z)," &
        "101 (BC_2,   *,          control, 1)," &          -- DISMISC
        "102 (BC_2,   *,          control, 1)," &          -- DISBUS
        "103 (BC_2,   *,          control, 1)," &          -- DISABUS
        "104 (BC_2,   *,          control, 1)";            -- DISWR

end Intel486TM_DX2;
```

241245-A1

Intel486™ DX2 CPU BSDL Model for Boundary Scan (Continued)

SEE INTEL PRELIMINARY FOR DESIGN-IN INFORMATION

9.0 DEBUGGING SUPPORT

The Intel486 microprocessor family provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

1) the code execution breakpoint opcode (0CCH),
2) the single-step capability provided by the TF bit in the flag register, and
3) the code and data breakpoint capability provided by the Debug Registers DR0–3, DR6, and DR7.

9.1 Breakpoint Instruction

A single-byte-opcode breakpoint instruction is available for use by software debuggers. The breakpoint opcode is 0CCH, and generates an exception 3 trap when executed. In typical use, a debugger program can "plant" the breakpoint instruction at all desired code execution breakpoints. The single-byte breakpoint opcode is an alias for the two-byte general software interrupt instruction, INT n, where n=3. The only difference between INT 3 (0CCh) and INT n is that INT 3 is never IOPL-sensitive but INT n is IOPL-sensitive in Protected Mode and Virtual 8086 Mode.

9.2 Single-Step Trap

If the single-step flag (TF, bit 8) in the EFLAG register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto vectored to exception number 1. Precisely, exception 1 occurs as a trap after the instruction following the instruction which set TF. In typical practice, a debugger sets the TF bit of a flag register image on the debugger's stack. It then typically transfers control to the user program and loads the flag image with a signal instruction, the IRET instruction. The single-step trap occurs after executing one instruction of the user program.

Since the exception 1 occurs as a trap (that is, it occurs after the instruction has already executed), the CS:EIP pushed onto the debugger's stack points to the next unexecuted instruction of the program being debugged. An exception 1 handler, merely by ending with an IRET instruction, can therefore efficiently support single-stepping through a user program.

9.3 Debug Registers

The Debug Registers are an advanced debugging feature of the Intel486 microprocessor family. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution break-point can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT3 breakpoint opcode.

The Intel486 microprocessor contains six Debug Registers, providing the ability to specify up to four distinct breakpoints addresses, breakpoint control options, and read breakpoint status. Initially after reset, breakpoints are in the disabled state. Therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are autovectored to exception number 1.

9.3.1 LINEAR ADDRESS BREAKPOINT REGISTERS (DR0–DR3)

Up to four breakpoint addresses can be specified by writing into Debug Registers DR0–DR3, shown in Figure 9.1. The breakpoint addresses specified are 32-bit linear addresses. Intel486 microprocessor hardware continuously compares the linear breakpoint addresses in DR0–DR3 with the linear addresses generated by executing software (a linear address is the result of computing the effective address and adding the 32-bit segment base address). Note that if paging is not enabled the linear address equals the physical address. If paging is enabled, the linear address is translated to a physical 32-bit address by the on-chip paging unit. Regardless of whether paging is enabled or not, however, the breakpoint registers hold linear addresses.

9.3.2 DEBUG CONTROL REGISTER (DR7)

A Debug Control Register, DR7 shown in Figure 9.1, allows several debug control functions such as enabling the breakpoints and setting up other control options for the breakpoints. The fields within the Debug Control Register, DR7, are as follows:

LENi (breakpoint length specification bits)

A 2-bit LEN field exists for each of the four breakpoints. LEN specifies the length of the associated breakpoint field. The choices for data breakpoints are: 1 byte, 2 bytes, and 4 bytes. Instruction execution breakpoints must have a length of 1 (LENi = 00). Encoding of the LENi field is as follows:

31 ... 16	15 ... 0	
BREAKPOINT 0 LINEAR ADDRESS		DR0
BREAKPOINT 1 LINEAR ADDRESS		DR1
BREAKPOINT 2 LINEAR ADDRESS		DR2
BREAKPOINT 3 LINEAR ADDRESS		DR3
Intel reserved. Do not define.		DR4
Intel reserved. Do not define.		DR5
0	BT BS BD 0 0 0 0 0 0 0 0 0 B3 B2 B1 B0	DR6
LEN3 R3 W3 LEN2 R2 W2 LEN1 R1 W1 LEN0 R0 W0	0 0 GD 0 0 0 GE LE G3 L3 G2 L2 G1 L1 G0 L0	DR7

31 16 15 0

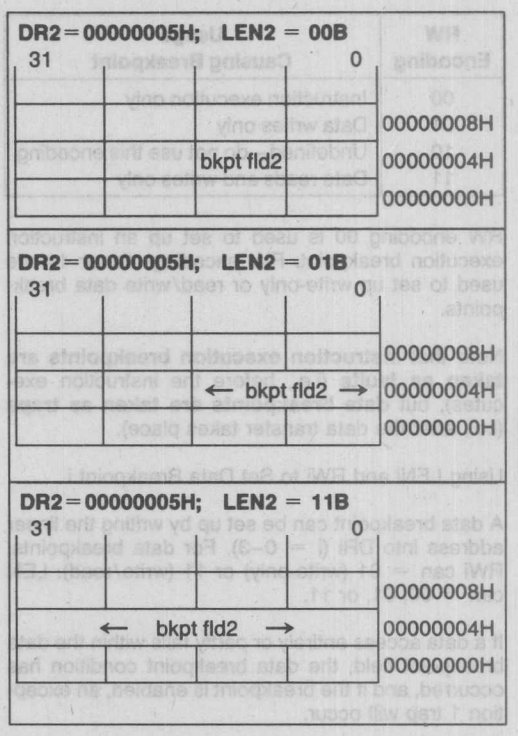
NOTE:
0 indicates Intel reserved: Do not define; SEE SECTION 2.3.10

Figure 9.1. Debug Registers

LENi Encoding	Breakpoint Field Width	Usage of Least Significant Bits in Breakpoint Address Register i, (i = 0 – 3)
00	1 byte	All 32-bits used to specify a single-byte breakpoint field.
01	2 bytes	A1–A31 used to specify a two-byte, word-aligned breakpoint field. A0 in Breakpoint Address Register is not used.
10	Undefined—do not use this encoding	
11	4 bytes	A2–A31 used to specify a four-byte, dword-aligned breakpoint field. A0 and A1 in Breakpoint Address Register are not used.

The LENi field controls the size of breakpoint field i by controlling whether all low-order linear address bits in the breakpoint address register are used to detect the breakpoint event. Therefore, all breakpoint fields are aligned; 2-byte breakpoint fields begin on Word boundaries, and 4-byte breakpoint fields begin on Dword boundaries.

The following is an example of various size breakpoint fields. Assume the breakpoint linear address in DR2 is 00000005H. In that situation, the following illustration indicates the region of the breakpoint field for lengths of 1, 2, or 4 bytes.



RWi (memory access qualifier bits)

A 2-bit RW field exists for each of the four breakpoints. The 2-bit RW field specifies the type of usage which must occur in order to activate the associated breakpoint.

2

RW Encoding	Usage Causing Breakpoint
00	Instruction execution only
01	Data writes only
10	Undefined—do not use this encoding
11	Data reads and writes only

RW encoding 00 is used to set up an instruction execution breakpoint. RW encodings 01 or 11 are used to set up write-only or read/write data breakpoints.

Note that **instruction execution breakpoints are taken as faults** (i.e., before the instruction executes), but **data breakpoints are taken as traps** (i.e., after the data transfer takes place).

Using LENi and RWi to Set Data Breakpoint i

A data breakpoint can be set up by writing the linear address into DRi (i = 0–3). For data breakpoints, RWi can = 01 (write-only) or 11 (write/read). LEN can = 00, 01, or 11.

If a data access entirely or partly falls within the data breakpoint field, the data breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 trap will occur.

Using LENi and RWi to Set Instruction Execution Breakpoint i

An instruction execution breakpoint can be set up by writing address of the beginning of the instruction (including prefixes if any) into DRi (i = 0–3). RWi must = 00 and LEN must = 00 for instruction execution breakpoints.

If the instruction beginning at the breakpoint address is about to be executed, the instruction execution breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 fault will occur before the instruction is executed.

Note that an instruction execution breakpoint address must be equal to the **beginning** byte address of an instruction (including prefixes) in order for the instruction execution breakpoint to occur.

GD (Global Debug Register access detect)

The Debug Registers can only be accessed in Real Mode or at privilege level 0 in Protected Mode. The GD bit, when set, provides extra protection against **any** Debug Register access even in Real Mode or at privilege level 0 in Protected Mode. This additional protection feature is provided to guarantee that a software debugger can have full control over the Debug Register resources when required. The GD bit, when set, causes an exception 1 fault if an instruction attempts to read or write any Debug Register. The GD bit is then automatically cleared when the exception 1 handler is invoked, allowing the exception 1 handler free access to the debug registers.

GE and LE (Exact data breakpoint match, global and local)

The breakpoint mechanism of the Intel486 microprocessor family differs from that of the Intel386. The Intel486 microprocessor always does exact data breakpoint matching, regardless of GE/LE bit settings. Any data breakpoint trap will be reported exactly after completion of the instruction that caused the operand transfer. Exact reporting is provided by forcing the Intel486 microprocessor execution unit to wait for completion of data operand transfers before beginning execution of the next instruction.

When the Intel486 microprocessor performs a task switch, the LE bit is cleared. Thus, the LE bit supports fast task switching out of tasks, that have enabled the exact data breakpoint match for their task-local breakpoints. The LE bit is cleared by the processor during a task switch, to avoid having exact data breakpoint match enabled in the new task. Note that exact data breakpoint match must be re-enabled under software control.

The Intel486 microprocessor GE bit is unaffected during a task switch. The GE bit supports exact data breakpoint match that is to remain enabled during all tasks executing in the system.

Note that **instruction execution** breakpoints are always reported exactly.

Gi and Li (breakpoint enable, global and local)

If either Gi or Li is set then the associated breakpoint (as defined by the linear address in DRi, the length in LENi and the usage criteria in RWi) is enabled. If either Gi or Li is set, and the Intel486 microprocessor detects the ith breakpoint condition, then the exception 1 handler is invoked.

When the Intel486 microprocessor performs a task switch to a new Task State Segment (TSS), all Li bits are cleared. Thus, the Li bits support fast task switching out of tasks that use some task-local breakpoint registers. The Li bits are cleared by the processor during a task switch, to avoid spurious exceptions in the new task. Note that the breakpoints must be re-enabled under software control.

All Intel486 microprocessor Gi bits are unaffected during a task switch. The Gi bits support breakpoints that are active in all tasks executing in the system.

PRELIMINARY

9.3.3 DEBUG STATUS REGISTER (DR6)

A Debug Status Register, DR6 shown in Figure 9.1, allows the exception 1 handler to easily determine why it was invoked. Note the exception 1 handler can be invoked as a result of one of several events:

1) DR0 Breakpoint fault/trap.
2) DR1 Breakpoint fault/trap.
3) DR2 Breakpoint fault/trap.
4) DR3 Breakpoint fault/trap.
5) Single-step (TF) trap.
6) Task switch trap.
7) Fault due to attempted debug register access when GD = 1.

The Debug Status Register contains single-bit flags for each of the possible events invoking exception 1. Note below that some of these events are faults (exception taken before the instruction is executed), while other events are traps (exception taken after the debug events occurred).

The flags in DR6 are set by the hardware but never cleared by hardware. Exception 1 handler software should clear DR6 before returning to the user program to avoid future confusion in identifying the source of exception 1.

The fields within the Debug Status Register, DR6, are as follows:

Bi (debug fault/trap due to breakpoint 0–3)

Four breakpoint indicator flags, B0–B3, correspond one-to-one with the breakpoint registers in DR0–DR3. A flag Bi is set when the condition described by DRi, LENi, and RWi occurs.

If Gi or Li is set, and if the ith breakpoint is detected, the processor will invoke the exception 1 handler. The exception is handled as a fault if an instruction execution breakpoint occurred, or as a trap if a data breakpoint occurred.

IMPORTANT NOTE: A flag Bi is set whenever the hardware detects a match condition on **enabled** breakpoint i. Whenever a match is detected on at least one **enabled** breakpoint i, the hardware immediately sets all Bi bits corresponding to breakpoint conditions matching at that instant, whether enabled **or not.** Therefore, the exception 1 handler may see that multiple Bi bits are set, but only set Bi bits corresponding to **enabled** breakpoints (Li or Gi set) are **true** indications of why the exception 1 handler was invoked.

BD (debug fault due to attempted register access when GD bit set)

This bit is set if the exception 1 handler was invoked due to an instruction attempting to read or write to the debug registers when GD bit was set. If such an event occurs, then the GD bit is automatically cleared when the exception 1 handler is invoked, allowing handler access to the debug registers.

BS (debug trap due to single-step)

This bit is set if the exception 1 handler was invoked due to the TF bit in the flag register being set (for single-stepping).

BT (debug trap due to task switch)

This bit is set if the exception 1 handler was invoked due to a task switch occurring to a task having a Intel486 microprocessor TSS with the T bit set. Note the task switch into the new task occurs normally, but before the first instruction of the task is executed, the exception 1 handler is invoked. With respect to the task switch operation, the operation is considered to be a trap.

9.3.4 USE OF RESUME FLAG (RF) IN FLAG REGISTER

The Resume Flag (RF) in the flag word can suppress an instruction execution breakpoint when the exception 1 handler returns to a user program at a user address which is also an instruction execution breakpoint.

10.0 INSTRUCTION SET SUMMARY

This section describes the Intel486 DX2 microprocessor instruction set. Tables 10.1 through 10.3 list all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in Section 10.2, which completely describes the encoding structure and the definition of all fields occurring within the Intel486 DX2 microprocessor instructions.

10.1 Intel486™ DX2 Microprocessor Instruction Encoding and Clock Count Summary

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Tables 10.1 through 10.3 by the processor core clock period (e.g., 20 ns for a 50 MHz Intel486 DX2 microprocessor).

For more detailed information on the encodings of instructions, refer to Section 10.2 Instruction Encodings. Section 10.2 explains the general structure of instruction encodings, and defines exactly the encodings of all fields contained within the instruction.

INSTRUCTION CLOCK COUNT ASSUMPTIONS

The Intel486 DX2 microprocessor instruction core clock count tables give clock counts assuming data and instruction accesses hit in the cache. The combined instruction and data cache hit rate is over 90%.

A cache miss will force the Intel486 DX2 microprocessor to run an external bus cycle. The Intel486 DX2 microprocessor 32-bit burst bus is defined as r−b−w.

Where:

r = The number of bus clocks in the first cycle of a burst read or the number of clocks per data cycle in a non-burst read.

b = The number of bus clocks for the second and subsequent cycles in a burst read.

w = The number of bus clocks for a write.

The fastest bus the Intel486 DX2 microprocessor can support is 2−1−2 assuming 0 wait states. The clock counts in the cache miss penalty column assume a 2−1−2 bus. For slower busses add r−2 clocks to the cache miss penalty for the first dword accessed. Other factors also affect instruction clock counts.

Instruction Clock Count Assumptions

1. The external bus is available for reads or writes at all times. Else add bus clocks to reads until the bus is available.
2. Accesses are aligned. Add three core clocks to each misaligned access.
3. Cache fills complete before subsequent accesses to the same line. If a read misses the cache during a cache fill due to a previous read or prefetch, the read must wait for the cache fill to complete. If a read or write accesses a cache line still being filled, it must wait for the fill to complete.
4. If an effective address is calculated, the base register is not the destination register of the preceding instruction. If the base register is the destination register of the preceding instruction add 1 to the core clock counts shown. Back-to-back PUSH and POP instructions are not affected by this rule.
5. An effective address calculation uses one base register and does not use an index register. However, if the effective address calculation uses an index register, 1 core clock **may** be added to the clock count shown.
6. The target of a jump is in the cache. If not, add r clocks for accessing the destination instruction of a jump. If the destination instruction is not completely contained in the first dword read, add a maximum of 3b bus clocks. If the destination instruction is not completely contained in the first 16 byte burst, add a maximum of another r+3b bus clocks.
7. If no write buffer delay, w bus clocks are added only in the case in which all write buffers are full.
8. Displacement and immediate not used together. If displacement and immediate used together, 1 core clock **may** be added to the core clock count shown.
9. No invalidate cycles. Add a delay of 1 bus clock for each invalidate cycle if the invalidate cycle contends for the internal cache/external bus when the Intel486 DX2 CPU needs to use it.
10. Page translation hits in TLB. A TLB miss will add 13, 21 or 28 bus clocks + 1 possible core clock to the instruction depending on whether the Accessed and/or Dirty bit in neither, one or both of the page entries needs to be set in memory. This assumes that neither page entry is in the data cache and a page fault does not occur on the address translation.
11. No exceptions are detected during instruction execution. Refer to Interrupt core Clock Counts Table for extra clocks if an interrupt is detected.
12. Instructions that read multiple consecutive data items (i.e. task switch, POPA, etc.) and miss the cache are assumed to start the first access on a 16-byte boundary. If not, an extra cache line fill may be necessary which may add up to (r+3b) bus clocks to the cache miss penalty.

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit	Notes
INTEGER OPERATIONS			
MOV = Move:			
reg1 to reg2	1000100W | 11 reg1 reg2	1	
reg2 to reg1	1000101w | 11 reg1 reg2	1	
memory to reg	1000101w | mod reg r/m	1	
reg to memory	1000100w | mod reg r/m	1	
Immediate to reg	1100011w | 11000 reg | immediate data	1	
or	1011w reg | immediate data	1	
Immediate to Memory	1100011w | mod 000 r/m | displacement immediate	1	
Memory to Accumulator	1010000w | full displacement	1	
Accumulator to Memory	1010001w | full displacement	1	
MOVSX/MOVZX = Move with Sign/Zero Extension			
reg2 to reg1	00001111 | 1011z11w | 11 reg1 reg2	3	
memory to reg	00001111 | 1011z11w | mod reg r/m	3	
PUSH = Push			
reg	11111111 | 11 110 reg	4	
or	01010 reg	1	
memory	11111111 | mod 110 r/m	4	
immediate	011010s0 | immediate data	1	
PUSHA = Push All	01100000	11	
POP = Pop			
reg	10001111 | 11 000 reg	4	
or	01011 reg	1	
memory	10001111 | mod 000 r/m	5	
POPA = Pop All	01100001	9	
XCHG = Exchange			
reg1 with reg2	1000011w | 11 reg1 reg2	3	
Accumulator with reg	10010 reg	3	
Memory with reg	1000011w | mod reg r/m	5	
NOP = No Operation	10010000	1	
LEA = Load EA to Register	10001101 | mod reg r/m		
no index register		1	
with index register		2	

z	instruction
0	MOVZX
1	MOVSX

2

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION FORMAT				Cache Hit	Notes
INTEGER OPERATIONS (Continued)					
Instruction	**TTT**				
ADD = Add	000				
ADC = Add with Carry	010				
AND = Logical AND	100				
OR = Logical OR	001				
SUB = Subtract	101				
SBB = Subtract with Borrow	011				
XOR = Logical Exclusive OR	110				
reg1 to reg2	00TTT00w	11 reg1 reg2		1	
reg2 to reg1	00TTT01w	11 reg1 reg2		1	
memory to register	00TTT01w	mod reg r/m		2	
register to memory	00TTT00w	mod reg r/m		3	U/L
immediate to register	100000sw	11 TTT reg	immediate register	1	
immediate to accumulator	00TTT10w	immediate data		1	
immediate to memory	100000sw	mod TTT r/m	immediate data	3	U/L
Instruction	**TTT**				
INC = Increment	000				
DEC = Decrement	001				
reg	1111111w	11 TTT reg		1	
or	01TTT reg			1	
memory	1111111w	mod TTT r/m		3	U/L
Instruction	**TTT**				
NOT = Logical Complement	010				
NEG = Negate	011				
reg	1111011w	11 TTT reg		1	
memory	1111011w	mod TTT r/m		3	U/L
CMP = Compare					
reg1 with reg2	0011100w	11 reg1 reg2		1	
reg2 with reg1	0011101w	11 reg1 reg2		1	
memory with register	0011100w	mod reg r/m		2	
register with memory	0011101w	mod reg r/m		2	
immediate with register	100000sw	11 111 reg	immediate data	1	
immediate with acc.	0011110w	immediate data		1	
immediate with memory	100000sw	mod 111 r/m	immediate data	2	
TEST = Logical Compare					
reg1 and reg2	1000010w	11 reg1 reg2		1	
memory and register	1000010w	mod reg r/m		2	
immediate and register	1111011w	11 000 reg	immediate data	1	
immediate and acc.	1010100w	immediate data		1	
immediate and memory	1111011w	mod 000 r/m	immediate data	2	

PRELIMINARY

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Notes
INTEGER OPERATIONS (Continued)			
MUL = Multiply (unsigned)			
acc. with register	1111011w | 11 100 reg		
Multiplier-Byte		13/18	MN/MX, 3
Word		13/26	MN/MX, 3
Dword		13/42	MN/MX, 3
acc. with memory	1111011w | mod 100 r/m		
Multiplier-Byte		13/18	MN/MX, 3
Word		13/26	MN/MX, 3
Dword		13/42	MN/MX, 3
IMUL = Integer Multiply (signed)			
acc. with register	1111011w | 11 101 reg		
Multiplier-Byte		13/18	MN/MX, 3
Word		13/26	MN/MX, 3
Dword		13/42	MN/MX, 3
acc. with memory	1111011w | mod 101 r/m		
Multiplier-Byte		13/18	MN/MX, 3
Word		13/26	MN/MX, 3
Dword		13/42	MN/MX, 3
reg1 with reg2	00001111 | 10101111 | 11 reg1 reg2		
Multiplier-Byte		13/18	MN/MX, 3
Word		13/26	MN/MX, 3
Dword		13/42	MN/MX, 3
register with memory	00001111 | 10101111 | mod reg r/m		
Multiplier-Byte		13/18	MN/MX, 3
Word		13/26	MN/MX, 3
Dword		13/42	MN/MX, 3
reg1 with imm. to reg2	011010s1 | 11 reg1 reg2 | immediate data		
Multiplier-Byte		13/18	MN/MX, 3
Word		13/26	MN/MX, 3
Dword		13/42	MN/MX, 3
mem. with imm. to reg.	011010s1 | mod reg r/m | immediate data		
Multiplier-Byte		13/18	MN/MX, 3
Word		13/26	MN/MX, 3
Dword		13/42	MN/MX, 3
DIV = Divide (unsigned)			
acc. by register	1111011w | 11 110 reg		
Divisor-Byte		16	
Word		24	
Dword		40	
acc. by memory	1111011w | mod 110 r/m		
Divisor-Byte		16	
Word		24	
Dword		40	
IDIV = Integer Divide (signed)			
acc. by register	1111011w | 11 111 reg		
Divisor-Byte		19	
Word		27	
Dword		43	

2

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Notes
INTEGER OPERATIONS (Continued)			
acc. by memory	1111011w mod111 r/m		
Divisor-Byte		20	
Word		28	
Dword		44	
CBW/CWDE = Convert Byte to Word/ Convert Word to Dword	10011000	3	
CWD/CDQ = Convert Word to Dword/ Convert Dword to Quadword	10011001	3	

Instruction	TTT
ROL = Rotate Left	000
ROR = Rotate Right	001
RCL = Rotate through Carry Left	010
RCR = Rotate through Carry Right	011
SHL/SAL = Shift Logical/Arithmetic Left	100
SHR = Shift Logical Right	101
SAR = Shift Arithmetic Right	111

Not Through Carry (ROL, ROR, SAL, SAR, SHL, and SHR)					
reg by 1	1101000w	11 TTT reg		3	
memory by 1	1101000w	mod TTT r/m		4	
reg by CL	1101001w	11 TTT reg		3	
memory by CL	1101001w	mod TTT r/m		4	
reg by immediate count	1100000w	11 TTT reg	immediate 8-bit data	2	
mem by immediate count	1100000w	mod TTT r/m	immediate 8-bit data	4	
Through Carry (RCL and RCR)					
reg by 1	1101000w	11 TTT reg		3	
memory by 1	1101000w	mod TTT r/m		4	
reg by CL	1101001w	11 TTT reg		8/30	MN/MX, 4
memory by CL	1101001w	mod TTT r/m		9/31	MN/MX, 5
reg by immediate count	1100000w	11 TTT reg	immediate 8-bit data	8/30	MN/MX, 4
mem by immediate count	1100000w	mod TTT r/m	immediate 8-bit data	9/31	MN/MX, 5

Instruction	TTT
SHLD = Shift Left Double	100
SHRD = Shift Right Double	101

register with immediate	00001111	10TTT100	11 reg2 reg1	imm 8-bit data	2
memory by immediate	00001111	10TTT100	mod reg r/m	imm 8-bit data	3
register by CL	00001111	10TTT101	11 reg2 reg1		3
memory by CL	00001111	10TTT101	mod reg r/m		4
BSWAP = Byte Swap	00001111	11001 reg			1
XADD = Exchange and Add					
reg1, reg2	00001111	1100000w	11 reg2 reg1		3
memory, reg	00001111	1100000w	mod reg r/m		4 U/L
CMPXCHG = Compare and Exchange					
reg1, reg2	00001111	1011000w	11 reg2 reg1		6
memory, reg	00001111	1011000w	mod reg r/m		7/10 6

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT			Cache Hit	Notes
CONTROL TRANSFER (within segment)					
NOTE: Times are jump taken/not taken					
Jccc = Jump on ccc					
8-bit displacement	0111tttn	8-bit disp.		3/1	T/NT, 23
full displacement	00001111	1000tttn	full displacement	3/1	T/NT, 23
NOTE: Times are jump taken/not taken					
SETcccc = Set Byte on cccc (Times are cccc true/false)					
reg	00001111	1001tttn	11 000 reg	4/3	
memory	00001111	1001tttn	mod 000 r/m	3/4	
LOOP = LOOP CX Times	11100010	8-bit disp.		7/6	L/NL, 23
LOOPZ/LOOPE = Loop with Zero/Equal	11100001	8-bit disp.		9/6	L/NL, 23
LOOPNZ/LOOPNE = Loop while Not Zero	11100000	8-bit disp.		9/6	L/NL, 23
JCXZ = Jump on CX Zero	11100011	8-bit disp.		8/5	T/NT, 23
JECXZ = Jump on ECX Zero	11100011	8-bit disp.		8/5	T/NT, 23
(Address Size Prefix Differentiates JCXZ for JECXZ)					
JMP = Unconditional Jump (within segment)					
Short	11101011	8-bit disp.		3	7, 23
Direct	11101001	full displacement		3	7, 23
Register Indirect	11111111	11 100 reg		5	7, 23
Memory Indirect	11111111	mod 100 r/m		5	7
CALL = Call (within segment)					
Direct	11101000	full displacement		3	7, 23
Register Indirect	11111111	11 010 reg		5	7, 23
Memory Indirect	11111111	mod 010 r/m		5	7
RET = Return from CALL (within segment)	11000011			5	
Adding Immediate to SP	11000010	16-bit disp.		5	

Mnemonic cccc	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less Than or Equal/Greater Than	1111

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Notes
CONTROL TRANSFER (within segment) (Continued)			
ENTER = Enter Procedure	11001000 | 16-bit disp., 8-bit level		
Level = 0		14	
Level = 1		17	
Level (L) > 1		17+3L	8
LEAVE = Leave Procedure	11001001	5	
MULTIPLE-SEGMENT INSTRUCTIONS			
MOV = Move			
reg. to segment reg.	10001110 | 11 sreg3 reg	3/9	RV/P, 9
memory to segment reg.	10001110 | mod sreg3 r/m	3/9	RV/P, 9
segment reg. to reg.	10001100 | 11 sreg3 reg	3	
segment reg. to memory	10001100 | mod sreg3 r/m	3	
PUSH = Push			
segment reg. (ES, CS, SS, or DS)	000sreg2110	3	
segment reg. (FS or GS)	00001111 | 10 sreg3000	3	
POP = Pop			
segment reg. (ES, SS, or DS)	000sreg2111	3/9	RV/P, 9
segment reg. (FS or GS)	00001111 | 10 sreg3001	3/9	RV/P, 9
LDS = Load Pointer to DS	11000101 | mod reg r/m	6/12	RV/P, 9
LES = Load Pointer to ES	11000100 | mod reg r/m	6/12	RV/P, 9
LFS = Load Pointer to FS	00001111 | 10110100 | mod reg r/m	6/12	RV/P, 9
LGS = Load Pointer to GS	00001111 | 10110101 | mod reg r/m	6/12	RV/P, 9
LSS = Load Pointer to SS	00001111 | 10110010 | mod reg r/m	6/12	RV/P, 9
CALL = Call			
Direct intersegment	10011010 | unsigned full offset, selector	18	R, 7, 22
to same level		20	P, 9
thru Gate to same level		35	P, 9
to inner level, no parameters		69	P, 9
to inner level, x parameter (d) words		77+4X	P, 11, 9
to TSS		37+TS	P, 10, 9
thru Task Gate		38+TS	P, 10, 9
Indirect intersegment	11111111 | mod 011 r/m	17	R, 7
to same level		20	P, 9
thru Gate to same level		35	P, 9
to inner level, no parameters		69	P, 9
to inner level, x parameter (d) words		77+4X+n	P, 11, 9
to TSS		37+TS	P, 10, 9
thru Task Gate		38+TS	P, 10, 9
RET = Return from CALL			
intersegment	11001011	13	R, 7
to same level		17	P, 9
to outer level		35	P, 9
intersegment adding imm. to SP	11001010 | 16-bit disp.	14	R, 7
to same level		18	P, 9
to outer level		36	P, 9

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Notes
MULTIPLE-SEGMENT INSTRUCTIONS (Continued)			
JMP = Unconditional Jump			
Direct intersegment	11101010 | unsigned full offset, selector	17	R, 7, 22
to same level		19	P, 9
thru Call Gate to same level		32	P, 9
thru TSS		42+TS	P, 10, 9
thru Task Gate		43+TS	P, 10, 9
Indirect intersegment	11111111 | mod 101 r/m	13	R, 7, 9
to same level		18	P, 9
thru Call Gate to same level		31	P, 9
thru TSS		41+TS	P, 10, 9
thru Task Gate		42+TS	P, 10, 9
BIT MANIPULATION			
BT = Test bit			
register, immediate	00001111 | 10111010 | 11 100 reg | imm. 8-bit data	3	
memory, immediate	00001111 | 10111010 | mod 100 r/m | imm. 8-bit data	3	
reg1, reg2	00001111 | 10100011 | 11 reg2 reg1	3	
memory, reg	00001111 | 10100011 | mod reg r/m	8	

Instruction	TTT
BTS = Test Bit and Set	101
BTR = Test Bit and Reset	110
BTC = Test Bit and Compliment	111

register, immediate	00001111	10111010	11 TTT reg	imm. 8-bit data	6	
memory, immediate	00001111	10111010	mod TTT r/m	imm. 8-bit data	8	U/L
reg1, reg2	00001111	10TTT011	11 reg2 reg1		6	
memory, reg	00001111	10TTT011	mod reg r/m		13	U/L
BSF = Scan Bit Forward						
reg1, reg2	00001111	10111100	11 reg2 reg1		6/42	MN/MX, 12
memory, reg	00001111	10111100	mod reg r/m		7/43	MN/MX, 13
BSR = Scan Bit Reverse						
reg1, reg2	00001111	10111101	11 reg2 reg1		6/103	MN/MX, 14
memory, reg	00001111	10111101	mod reg r/m		7/104	MN/MX, 15
STRING INSTRUCTIONS						
CMPS = Compare Byte Word	1010011w				8	16
LODS = Load Byte/Word to AL/AX/EAX	1010110w				5	
MOVS = Move Byte/Word	1010010w				7	16
SCAS = Scan Byte/Word	1010111w				6	
STOS = Store Byte/Word from AL/AX/EX	1010101w				5	
XLAT = Translate String	11010111				4	

2

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT		Cache Hit	Notes
REPEATED STRING INSTRUCTIONS				
Repeated by Count in CX or ECX (C = Count in CX or ECX)				
REPE CMPS = Compare String (Find Non-Match)	11110011	1010011w		
C = 0			5	
C > 0			7+7c	16, 17
REPNE CMPS = Compare String (Find Match)	11110010	1010011w		
C = 0			5	
C > 0			7+7c	16, 17
REP LODS = Load String	11110011	1010110w		
C = 0			5	
C > 0			7+4c	16, 18
REP MOVS = Move String	11110011	1010010w		
C = 0			5	
C = 1			13	16
C > 1			12+3c	16, 19
REPE SCAS = Scan String (Find Non-AL/AX/EAX)	11110011	1010111w		
C = 0			5	
C > 0			7+5c	20
REPNE SCAS = Scan String (Find AL/AX/EAX)	11110010	1010111w		
C = 0			5	
C > 0			7+5c	20
REP STOS = Store String	11110011	1010101w		
C = 0			5	
C > 0			7+4c	
FLAG CONTROL				
CLC = Clear Carry Flag	11111000		2	
STC = Set Carry Flag	11111001		2	
CMC = Complement Carry Flag	11110101		2	
CLD = Clear Direction Flag	11111100		2	
STD = Set Direction Flag	11111101		2	
CLI = Clear Interrupt Enable Flag	11111010		5	
STI = Set Interrupt Enable Flag	11111011		5	
LAHF = Load AH into Flag	10011111		3	
SAHF = Store AH into Flags	10011110		2	
PUSHF = Push Flags	10011100		4/3	RV/P
POPF = Pop Flags	10011101		9/6	RV/P
DECIMAL ARITHMETIC				
AAA = ASCII Adjust for Add	00110111		3	
AAS = ASCII Adjust for Subtract	00111111		3	
AAM = ASCII Adjust for Multiply	11010100	00001010	15	

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Notes
DECIMAL ARITHMETIC (Continued)			
AAD = ASCII Adjust for Divide	11010101 00001010	14	
DAA = Decimal Adjust for Add	00100111	2	
DAS = Decimal Adjust for Subtract	00101111	2	
PROCESSOR CONTROL INSTRUCTIONS			
HLT = Halt	11110100	4	
MOV = Move To and From Control/Debug/Test Registers			
CR0 from register	00001111 00100010 11 000 reg	17	
CR2/CR3 from register	00001111 00100010 11 eee reg	4	
Reg from CR0–3	00001111 00100000 11 eee reg	4	
DR0–3 from register	00001111 00100011 11 eee reg	10	
DR6–7 from register	00001111 00100011 11 eee reg	10	
Register from DR6–7	00001111 00100001 11 eee reg	9	
Register from DR0–3	00001111 00100001 11 eee reg	9	
TR3 from register	00001111 00100110 11 011 reg	4	
TR4–7 from register	00001111 00100110 11 eee reg	4	
Register from TR3	00001111 00100100 11 011 reg	3	
Register from TR4–7	00001111 00100100 11 eee reg	4	
CLTS = Clear Task Switched Flag	00001111 00000110	7	
INVD = Invalidate Data Cache	00001111 00001000	4	
WBINVD = Write-Back and Invalidate Data Cache	00001111 00001001	5	
INVLPG = Invalidate TLB Entry			
INVLPG memory	00001111 00000001 mod 111 r/m	12/11	H/NH
PREFIX BYTES			
Address Size Prefix	01100111	1	
LOCK = Bus Lock Prefix	11110000	1	
Operand Size Prefix	01100110	1	
Segment Override Prefix			
CS:	00101110	1	
DS:	00111110	1	
ES:	00100110	1	
FS:	01100100	1	
GS:	01100101	1	
SS:	00110110	1	

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT			Cache Hit	Notes
PROTECTION CONTROL					
ARPL = Adjust Requested Privilege Level					
From register	01100011	11 reg1 reg2		9	
From memory	01100011	mod reg r/m		9	
LAR = Load Access Rights					
From register	00001111	00000010	11 reg1 reg2	11	
From memory	00001111	00000010	mod reg r/m	11	
LGDT = Load Global Descriptor					
Table register	00001111	00000001	mod 010 r/m	12	
LIDT = Load Interrupt Descriptor					
Table register	00001111	00000001	mod 011 r/m	12	
LLDT = Load Local Descriptor					
Table register from reg.	00001111	00000000	11 010 reg	11	
Table register from mem.	00001111	00000000	mod 010 r/m	11	
LMSW = Load Machine Status Word					
From register	00001111	00000001	11 110 reg	13	
From memory	00001111	00000001	mod 110 r/m	13	
LSL = Load Segment Limit					
From register	00001111	00000011	11 reg1 reg2	10	
From memory	00001111	00000011	mod reg r/m	10	
LTR = Load Task Register					
From Register	00001111	00000000	11 011 reg	20	
From Memory	00001111	00000000	mod 011 r/m	20	
SGDT = Store Global Descriptor Table					
	00001111	00000001	mod 000 r/m	10	
SIDT = Store Interrupt Descriptor Table					
	00001111	00000001	mod 001 r/m	10	
SLDT = Store Local Descriptor Table					
To register	00001111	00000000	11 000 reg	2	
To memory	00001111	00000000	mod 000 r/m	3	
SMSW = Store Machine Status Word					
To register	00001111	00000001	11 100 reg	2	
To memory	00001111	00000001	mod 100 r/m	3	
STR = Store Task Register					
To register	00001111	00000000	11 001 reg	2	
To memory	00001111	00000000	mod 001 r/m	3	
VERR = Verify Read Access					
Register	00001111	00000000	11 100 r/m	11	
Memory	00001111	00000000	mod 100 r/m	11	
VERW = Verify Write Access					
To register	00001111	00000000	11 101 reg	11	
To memory	00001111	00000000	mod 101 r/m	11	

PRELIMINARY

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Notes
INTERRUPT INSTRUCTIONS			
INT n = Interrupt Type n	11001101 | type	INT+4/0	RV/P, 21
INT 3 = Interrupt Type 3	11001100	INT+0	21
INTO = Interrupt 4 if Overflow Flag Set	11001110		
Taken		INT+2	21
Not Taken		3	21
BOUND = Interrupt 5 if Detect Value Out Range	01100010 | mod reg r/m		
If in range		7	21
If out of range		INT+24	21
IRET = Interrupt Return	11001111		
Real Mode/Virtual Mode		15	
Protected Mode			
To same level		20	9
To outer level		36	9
To nested task (EFLAGS.NT = 1)		TS+32	9, 10
External Interrupt		INT+11	21
NMI = Non-Maskable Interrupt		INT+3	21
Page Fault		INT+24	21
VM86 Exceptions			
CLI		INT+8	21
STI		INT+8	21
INT n		INT+9	
PUSHF		INT+9	21
POPF		INT+8	21
IRET		INT+9	
IN			
Fixed Port		INT+50	21
Variable Port		INT+51	21
OUT			
Fixed Port		INT+50	21
Variable Port		INT+51	21
INS		INT+50	21
OUTS		INT+50	21
REP INS		INT+51	21
REP OUTS		INT+51	21

Task Switch Clock Counts Table	
Method	**Value for TS Cache Hit**
VM/Intel486 DX2 CPU/286 TSS To Intel486 DX2 CPU TSS	162
VM/Intel486 DX2 CPU/286 TSS To 286 TSS	143
VM/Intel486 DX2 CPU/286 TSS To VM TSS	140

2

Interrupt Clock Counts Table

Method	Value for INT	
	Cache Hit	Notes
Real Mode	26	
Protected Mode		
Interrupt/Trap gate, same level	44	9
Interrupt/Trap gate, different level	71	9
Task Gate	37 + TS	9, 10
Virtual Mode		
Interrupt/Trap gate, different level	82	
Task gate	37 + TS	10

Abbreviations	Definition
16/32	16/32 bit modes
U/L	unlocked/locked
MN/MX	minimum/maximum
L/NL	loop/no loop
RV/P	real and virtual mode/protected mode
R	real mode
P	protected mode
T/NT	taken/not taken
H/NH	hit/no hit

NOTES:

1. Assuming that the operand address and stack address fall in different cache sets.
2. Always locked, no cache hit case.
3. Clocks = 10 + max($\log_2(|m|)$,n)
   m = multiplier value (min clocks for m=0)
   n = 3/5 for ±m
4. Clocks = {quotient(count/operand length)}*7+9
   = 8 if count ≤ operand length (8/16/32)
5. Clocks = {quotient(count/operand length)}*7+9
   = 9 if count ≤ operand length (8/16/32)
6. Equal/not equal cases (penalty is the same regardless of lock).
7. Assuming that addresses for memory read (for indirection), stack push/pop, and branch fall in different cache sets.
8. Penalty for cache miss: add 6 clocks for every 16 bytes copied to new stack frame.
9. Add 11 clocks for each unaccessed descriptor load.
10. Refer to task switch clock counts table for value of TS.
11. Add 4 extra clocks to the cache miss penalty for each 16 bytes.

For notes 12–13: (b = 0–3, non-zero byte number);
(i = 0–1, non-zero nibble number);
(n = 0–3, non bit number in nibble);

12. Clocks = 8+4 (b+1) + 3(i+1) + 3(n+1)
    = 6 if second operand = 0
13. Clocks = 9+4(b+1) + 3(i+1) + 3(n+1)
    = 7 if second operand = 0

For notes 14–15: (n = bit position 0–31)

14. Clocks = 7 + 3(32−n)
    6 if second operand = 0
15. Clocks = 8 + 3(32−n)
    7 if second operand = 0
16. Assuming that the two string addresses fall in different cache sets.
17. Cache miss penalty: add 6 clocks for every 16 bytes compared. Entire penalty on first compare.
18. Cache miss penalty: add 2 clocks for every 16 bytes of data. Entire penalty on first load.
19. Cache miss penalty: add 4 clocks for every 16 bytes moved.
    (1 clock for the first operation and 3 for the second)
20. Cache miss penalty: add 4 clocks for every 16 bytes scanned.
    (2 clocks each for first and second operations)
21. Refer to interrupt clock counts table for value of INT
22. Clock count includes one clock for using both displacement and immediate.
23. Refer to assumption 6 in the case of a cache miss.

Table 10.2. Intel486™ DX2 Microprocessor I/O Instructions Core Clock Count Summary

INSTRUCTION	FORMAT	Real Mode	Protected Mode (CPL≤IOPL)	Protected Mode (CPL>IOPL)	Virtual 86 Mode	Notes
I/O INSTRUCTIONS						
IN = Input from:						
Fixed Port	1110010w | port number	17	12	32	30	
Variable Port	1110110w	17	11	31	30	
OUT = Output to:						
Fixed Port	1110011w | port number	19	14	34	32	
Variable Port	1110111w	19	13	33	32	
INS = Input Byte/Word from DX Port	0110110w	20	13	35	33	
OUTS = Output Byte/Word to DX Port	0110111w	20	13	35	33	1
REP INS = Input String	11110011 | 0110110w	19+11c	13+11c	33+11c	32+11c	2
REP OUTS = Output String	11110011 | 0110111w	20+8c	14+8c	34+8c	33+8c	3

NOTES:
1. Two clock cache miss penalty in all cases.
2. c = count in CX or ECX.
3. Cache miss penalty in all modes: Add 2 clocks for every 16 bytes. Entire penalty on second operation.

2

Table 10.3. Intel486™ DX2 Microprocessor Floating Point Core Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range . . . Upper Range)	Notes
DATA TRANSFER			
FLD = Real Load to ST(0)			
32-bit memory	11011 001 | mod 000 r/m | s-i-b/disp.	3	
64-bit memory	11011 101 | mod 000 r/m | s-i-b/disp.	3	
80-bit memory	11011 011 | mod 101 r/m | s-i-b/disp.	6	
ST(i)	11011 001 | 11000 ST(i)	4	
FILD = Integer Load to ST(0)			
16-bit memory	11011 111 | mod 000 r/m | s-i-b/disp.	14.5(13–16)	
32-bit memory	11011 011 | mod 000 r/m | s-i-b/disp.	11.5(9–12)	
64-bit memory	11011 111 | mod 101 r/m | s-i-b/disp.	16.8(10–18)	
FBLD = BCD Load to ST(0)	11011 111 | mod 100 r/m | s-i-b/disp.	75(70–103)	
FST = Store Real from ST(0)			
32-bit memory	11011 001 | mod 010 r/m | s-i-b/disp.	7	1
64-bit memory	11011 101 | mod 010 r/m | s-i-b/disp.	8	2
ST(i)	11011 101 | 11010 ST(i)	3	
FSTP = Store Real from ST(0) and Pop			
32-bit memory	11011 001 | mod 011 r/m | s-i-b/disp.	7	1
64-bit memory	11011 101 | mod 011 r/m | s-i-b/disp.	8	2
80-bit memory	11011 011 | mod 111 r/m | s-i-b/disp.	6	
ST(i)	11011 101 | 11001 ST(i)	3	
FIST = Store Integer from ST(0)			
16-bit memory	11011 111 | mod 010 r/m | s-i-b/disp.	33.4(29–34)	
32-bit memory	11011 011 | mod 010 r/m | s-i-b/disp.	32.4(28–34)	
FISTP = Store Integer from ST(0) and Pop			
16-bit memory	11011 111 | mod 011 r/m | s-i-b/disp.	33.4(29–34)	
32-bit memory	11011 011 | mod 011 r/m | s-i-b/disp.	33.4(29–34)	
64-bit memory	11011 111 | mod 111 r/m | s-i-b/disp.	33.4(29–34)	
FBSTP = Store BCD from ST(0) and Pop	11011 111 | mod 110 r/m | s-i-b/disp.	175(172–176)	
FXCH = Exchange ST(0) and ST(i)	11011 001 | 11001 ST(i)	4	
COMPARISON INSTRUCTIONS			
FCOM = Compare ST(0) with Real			
32-bit memory	11011 000 | mod 010 r/m | s-i-b/disp.	4	
64-bit memory	11011 100 | mod 010 r/m | s-i-b/disp.	4	
ST(i)	11011 000 | 11010 ST(i)	4	
FCOMP = Compare ST(0) with Real and Pop			
32-bit memory	11011 000 | mod 011 r/m | s-i-b/disp.	4	
64-bit memory	11011 100 | mod 011 r/m | s-i-b/disp.	4	
ST(i)	11011 000 | 11011 ST(i)	4	

PRELIMINARY

Table 10.3. Intel486™ DX2 Microprocessor Floating Point Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range . . . Upper Range)	Notes
COMPARISON INSTRUCTIONS (Continued)			
FCOMPP = Compare ST(0) with ST(1) and Pop Twice	11011 110 | 1101 1001	5	
FICOM = Compare ST(0) with Integer			
16-bit memory	11011 110 | mod 010 r/m | s-i-b/disp.	18(16–20)	
32-bit memory	11011 010 | mod 010 r/m | s-i-b/disp.	16.5(15–17)	
FICOMP = Compare ST(0) with Integer			
16-bit memory	11011 110 | mod 011 r/m | s-i-b/disp.	18(16–20)	
32-bit memory	11011 010 | mod 011 r/m | s-i-b/disp.	16.5(15–17)	
FTST = Compare ST(0) with 0.0	11011 001 | 1110 0100	4	
FUCOM = Unordered compare ST(0) with ST(i)	11011 101 | 11100 ST(i)	4	
FUCOMP = Unordered compare ST(0) with ST(i) and Pop	11011 101 | 11101 ST(i)	4	
FUCOMPP = Unordered compare ST(0) with ST(i) and Pop Twice	11011 010 | 1110 1001	5	
FXAM = Examine ST(0)	11011 001 | 1110 0101	8	
CONSTANTS			
FLDZ = Load +0.0 into ST(0)	11011 001 | 1110 1110	4	
FLD1 = Load +1.0 into ST(0)	11011 001 | 1110 1000	4	
FLDPI = Load π into ST(0)	11011 001 | 1110 1011	8	
FLDL2T = Load $\log_2(10)$ into ST(0)	11011 001 | 1110 1001	8	
FLDL2E = Load $\log_2(e)$ into ST(0)	11011 001 | 1110 1010	8	
FLDLG2 = Load $\log_{10}(2)$ into ST(0)	11011 001 | 1110 1100	8	
FLDLN2 = Load $\log_e(2)$ into ST(0)	11011 001 | 1110 1101	8	
ARITHMETIC			
FADD = Add Real with ST(0)			
ST(0) ← ST(0) + 32-bit memory	11011 000 | mod 000 r/m | s-i-b/disp.	10(8–20)	
ST(0) ← ST(0) + 64-bit memory	11011 100 | mod 000 r/m | s-i-b/disp.	10(8–20)	
ST(d) ← ST(0) + ST(i)	11011 d00 | 11000 ST(i)	10(8–20)	
FADDP = Add real with ST(0) and Pop (ST(i) ← ST(0) + ST(i))	11011 110 | 11000 ST(i)	10(8–20)	
FSUB = Subtract real from ST(0)			
ST(0) ← ST(0) − 32-bit memory	11011 000 | mod 100 r/m | s-i-b/disp.	10(8–20)	
ST(0) ← ST(0) − 64-bit memory	11011 100 | mod 100 r/m | s-i-b/disp.	10(8-20)	
ST(d) ← ST(0) − ST(i)	11011 d00 | 1110d ST(i)	10(8–20)	
FSUBP = Subtract real from ST(0) and Pop (ST(i) ← ST(0) − ST(i))	11011 110 | 11101 ST(i)	10(8–20)	

Table 10.3. Intel486™ DX2 Microprocessor Floating Point Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range . . . Upper Range)	Notes
ARITHMETIC (Continued)			
FSUBR = Subtract real reversed (Subtract ST(0) from real)			
ST(0) ← 32-bit memory − ST(0)	11011 000 | mod 101 r/m | s-i-b/disp.	10(8–20)	
ST(0) ← 64-bit memory − ST(0)	11011 100 | mod 101 r/m | s-i-b/disp.	10(8–20)	
ST(d) ← ST(i) − ST(0)	11011 d00 | 1110d ST(i)	10(8–20)	
FSUBRP = Subtract real reversed and Pop (ST(i) ← ST(i) − ST(0))	11011 110 | 11100 ST(i)	10(8–20)	
FMUL = Multiply real with ST(0)			
ST(0) ← ST(0) × 32-bit memory	11011 000 | mod 001 r/m | s-i-b/disp.	11	
ST(0) ← ST(0) × 64-bit memory	11011 100 | mod 001 r/m | s-i-b/disp.	14	
ST(d) ← ST(0) × ST(i)	11011 d00 | 11001 ST(i)	16	
FMULP = Multiply ST(0) with ST(i) and Pop (ST(i) ← ST(0) × ST(i))	11011 110 | 11001 ST(i)	16	
FDIV = Divide ST(0) by Real			
ST(0) ← ST(0)/32-bit memory	11011 000 | mod 110 r/m | s-i-b/disp.	73	3
ST(0) ← ST(0)/64-bit memory	11011 100 | mod 110 r/m | s-i-b/disp.	73	3
ST(d) ← ST(0)/ST(i)	11011 d00 | 1111d ST(i)	73	3
FDIVP = Divide ST(0) by ST(i) and Pop (ST(i) ← ST(0)/ST(i))	11011 110 | 11111 ST(i)	73	3
FDIVR = Divide real reversed (Real/ST(0))			
ST(0) ← 32-bit memory/ST(0)	11011 000 | mod 111 r/m | s-i-b/disp.	73	3
ST(0) ← 64-bit memory/ST(0)	11011 100 | mod 111 r/m | s-i-b/disp.	73	3
ST(d) ← ST(i)/ST(0)	11011 d00 | 1111d ST(i)	73	3
FDIVRP = Divide real reversed and Pop (ST(i) ← ST(i)/ST(0))	11011 110 | 11110 ST(i)	73	3
FIADD = Add Integer to ST(0)			
ST(0) ← ST(0) + 16-bit memory	11011 110 | mod 000 r/m | s-i-b/disp.	24(20–35)	
ST(0) ← ST(0) + 32-bit memory	11011 010 | mod 000 r/m | s-i-b/disp.	22.5(19–32)	
FISUB = Subtract Integer from ST(0)			
ST(0) ← ST(0) − 16-bit memory	11011 110 | mod 100 r/m | s-i-b/disp.	24(20–35)	
ST(0) ← ST(0) − 32-bit memory	11011 010 | mod 100 r/m | s-i-b/disp.	22.5(19–32)	
FISUBR = Integer Subtract Reversed			
ST(0) ← 16-bit memory − ST(0)	11011 110 | mod 101 r/m | s-i-b/disp.	24(20–35)	
ST(0) ← 32-bit memory − ST(0)	11011 010 | mod 101 r/m | s-i-b/disp.	22.5(19–32)	
FIMUL = Multiply Integer with ST(0)			
ST(0) ← ST(0) × 16-bit memory	11011 110 | mod 001 r/m | s-i-b/disp.	25(23–27)	
ST(0) ← ST(0) × 32-bit memory	11011 010 | mod 001 r/m | s-i-b/disp.	23.5(22–24)	
FIDIV = Integer Divide			
ST(0) ← ST(0)/16-bit memory	11011 110 | mod 110 r/m | s-i-b/disp.	87(85–89)	3
ST(0) ← ST(0)/32-bit memory	11011 010 | mod 110 r/m | s-i-b/disp.	85.5(84–86)	3

Table 10.3. Intel486™ DX2 Microprocessor Floating Point Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range ... Upper Range)	Notes
ARITHMETIC (Continued)			
FIDIVR = Integer Divide Reversed			
ST(0) ← 16-bit memory/ST(0)	11011 110 | mod 111 r/m | s-i-b/disp.	87(85–89)	3
ST(0) ← 32-bit memory/ST(0)	11011 010 | mod 111 r/m | s-i-b/disp.	85.5(84–86)	3
FSQRT = Square Root	11011 001 | 1111 1010	85.5(83–87)	
FSCALE = Scale ST(0) by ST(1)	11011 001 | 1111 1101	31(30–32)	
FXTRACT = Extract components of ST(0)	11011 001 | 1111 0100	19(16–20)	
FPREM = Partial Reminder	11011 001 | 1111 1000	84(70–138)	
FPREM1 = Partial Reminder (IEEE)	11011 001 | 1111 0101	94.5(72–167)	
FRNDINT = Round ST(0) to integer	11011 001 | 1111 1100	29.1(21–30)	
FABS = Absolute value of ST(0)	11011 001 | 1110 0001	3	
FCHS = Change sign of ST(0)	11011 001 | 1110 0000	6	
TRANSCENDENTAL			
FCOS = Cosine of ST(0)	11011 001 | 1111 1111	241(193–279)	6, 7
FPTAN = Partial tangent of ST(0)	11011 001 | 1111 0010	244(200–273)	6, 7
FPATAN = Partial arctangent	11011 001 | 1111 0011	289(218–303)	6
FSIN = Sine of ST(0)	11011 001 | 1111 1110	241(193–279)	6, 7
FSINCOS = Sine and cosine of ST(0)	11011 001 | 1111 1011	291(243–329)	6, 7
F2XM1 = $2^{ST(0)} - 1$	11011 001 | 1111 0000	242(140–279)	6
FYL2X = ST(1) × $\log_2$(ST(0))	11011 001 | 1111 0001	311(196–329)	6
FYL2XP1 = ST(1) × $\log_2$(ST(0) + 1.0)	11011 001 | 1111 1001	313(171–326)	6
PROCESSOR CONTROL			
FINIT = Initialize FPU	11011 011 | 1110 0011	17	4
FSTSW AX = Store status word into AX	11011 111 | 1110 0000	3	5
FSTSW = Store status word into memory	11011 101 | mod 111 r/m | s-i-b/disp.	3	5
FLDCW = Load control word	11011 001 | mod 101 r/m | s-i-b/disp.	4	
FSTCW = Store control word	11011 001 | mod 111 r/m | s-i-b/disp.	3	5
FCLEX = Clear exceptions	11011 011 | 1110 0010	7	4
FSTENV = Store environment	11011 001 | mod 110 r/m | s-i-b/disp.		
Real and Virtual modes 16-bit Address		67	4
Real and Virtual modes 32-bit Address		67	4
Protected mode 16-bit Address		56	4
Protected mode 32-bit Address		56	4
FLDENV = Load environment	11011 001 | mod 100 r/m | s-i-b/disp.		
Real and Virtual modes 16-bit Address		44	
Real and Virtual modes 32-bit Address		44	
Protected mode 16-bit Address		34	
Protected mode 32-bit Address		34	

2

Table 10.3. Intel486™ DX2 Microprocessor Floating Point Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range . . . Upper Range)	Notes
PROCESSOR CONTROL (Continued)			
FSAVE = Save state	11011 101 | mod 110 r/m | s-i-b/disp.		
Real and Virtual modes 16-bit Address		154	4
Real and Virtual modes 32-bit Address		154	4
Protected mode 16-bit Address		143	4
Protected mode 32-bit Address		143	4
FRSTOR = Restore state	11011 101 | mod 100 r/m | s-i-b/		
Real and Virtual modes 16-bit Address		131	
Real and Virtual modes 32-bit Address		131	
Protected mode 16-bit Address		120	
Protected mode 32-bit Address		120	
FINCSTP = Increment Stack Pointer	11011 001 | 1111 0111	3	
FDECSTP = Decrement Stack Pointer	11011 001 | 1111 0110	3	
FFREE = Free ST(i)	11011 101 | 11000 ST(i)	3	
FNOP = No operations	11011 001 | 1101 0000	3	
WAIT = Wait until FPU ready	10011011		
(Minimum/Maximum)		1/3	

NOTES:
1. If operand is 0 clock counts = 27.
2. If operand is 0 clock counts = 28.
3. If CW.PC indicates 24 bit precision then subtract 38 clocks.
   If CW.PC indicates 53 bit precision then subtract 11 clocks.
4. If there is a numeric error pending from a previous instruction add 17 clocks.
5. If there is a numeric error pending from a previous instruction add 18 clocks.
6. The INT pin is polled several times while this instruction is executing to assure short interrupt latency.
7. If ABS(operand) is greater than $\pi/4$ then add n clocks. Where n = (operand/($\pi/4$)).

10.2 Instruction Encoding

10.2.1 OVERVIEW

All instruction encodings are subsets of the general instruction format shown in Figure 10.1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the "mod r/m" byte and "scaled index" byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16 or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 10.1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the Figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 10.4 is a complete list of all fields appearing in the Intel486 DX2 microprocessor instruction set. Further ahead, following Table 10.4, are detailed tables for each field.

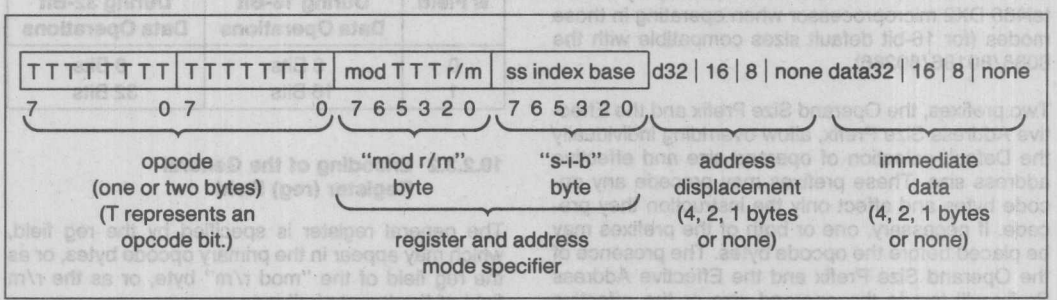


Figure 10.1. General Instruction Format

Table 10.4. Fields within Intel486™ DX2 Microprocessor Instructions

Field Name	Description	Number of Bits
w	Specifies if Data is Byte or Full Size (Full Size is either 16 or 32 Bits)	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field Must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
tttn	For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated	4

NOTE:
Tables 10.1–10.3 show encoding of individual instructions.

10.2.2 32-BIT EXTENSIONS OF THE INSTRUCTION SET

The Intel486 DX2 supports all Intel486 extensions to the 8086/80186/80286 instruction set.

With the Intel486 microprocessor, the 8086/80186/80286 instruction set was extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

Whether the instruction defaults to operations of 16 bits or 32 bits depends on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 bits or 16 bits) for both operands and effective addresses when executing that code segment. In the Real Address Mode or Virtual 8086 Mode, no code segment descriptors are used, but a D value of 0 is assumed internally by the Intel486 DX2 microprocessor when operating in those modes (for 16-bit default sizes compatible with the 8086/80186/80286).

Two prefixes, the Operand Size Prefix and the Effective Address Size Prefix, allow overriding individually the Default selection of operand size and effective address size. These prefixes may precede any opcode bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed before the opcode bytes. The presence of the Operand Size Prefix and the Effective Address Prefix will toggle the operand size or the effective address size, respectively, to the value "opposite" from the Default setting. For example, if the default operand size is for 32-bit data operations, then presence of the Operand Size Prefix toggles the instruction to 16-bit data operation. As another example, if the default effective address size is 16 bits, presence of the Effective Address Size prefix toggles the instruction to use 32-bit effective address computations.

These 32-bit extensions are available in all Intel486 microprocessor modes, including the Real Address Mode or the Virtual 8086 Mode. In these modes the default is always 16 bits, so prefixes are needed to specify 32-bit operands or addresses. For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

10.2.3 ENCODING OF INTEGER INSTRUCTION FIELDS

Within the instruction are several fields indicating register selection, addressing mode and so on. The exact encodings of these fields are defined immediately ahead.

10.2.3.1 Encoding of Operand Length (w) Field

For any given instruction performing a data operation, the instruction is executing as a 32-bit operation or a 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size During 16-Bit Data Operations	Operand Size During 32-Bit Data Operations
0	8 Bits	8 Bits
1	16 Bits	32 Bits

10.2.3.2 Encoding of the General Register (reg) Field

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the "mod r/m" byte, or as the r/m field of the "mod r/m" byte.

Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected During 16-Bit Data Operations	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations:		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field During 32-Bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

10.2.3.3 Encoding of the Segment Register (sreg) Field

The sreg field in certain instructions is a 2-bit field allowing one of the four 80286 segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the Intel486 DX2 Microprocessor FS and GS segment registers to be specified.

2-Bit sreg2 Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

10.2.3.4 Encoding of Address Mode

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the "mod r/m" byte, and a second byte of addressing information, the "s-i-b" (scale-index-base) byte, can be specified.

The s-i-b byte (scale-index-base byte) is specified when using 32-bit addressing mode and the "mod r/m" byte has r/m = 100 and mod = 00, 01 or 10. When the sib byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the "mod r/m" byte, also contains three bits (shown as TTT in Figure 10.1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the "mod r/m" byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the "mod r/m" byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following three pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Encoding of 16-bit Address Mode with "mod r/m" Byte

mod r/m	Effective Address
00 000	DS:[BX + SI]
00 001	DS:[BX + DI]
00 010	SS:[BP + SI]
00 011	SS:[BP + DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BP + DI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by r/m During 16-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m During 32-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode with "mod r/m" byte (no "s-i-b" byte present)

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by reg or r/m during 16-Bit Data Operations:		
mod r/m	Function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by reg or r/m during 32-Bit Data Operations:		
mod r/m	Function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode ("mod r/m" byte and "s-i-b" byte present)

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

NOTE:
Mod field in "mod r/m" byte; ss, index, base fields in "s-i-b" byte.

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg**
101	EBP
110	ESI
111	EDI

****IMPORTANT NOTE:**
When index field is 100, indicating "no index register," then ss field MUST equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

10.2.3.5 Encoding of Operation Direction (d) Field

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory <- - Register "reg" Field Indicates Source Operand; "mod r/m" or "mod ss index base" Indicates Destination Operand
1	Register <- - Register/Memory "reg" Field Indicates Destination Operand; "mod r/m" or "mod ss index base" Indicates Source Operand

10.2.3.6 Encoding of Sign-Extend (s) Field

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

s	Effect on Immediate Data8	Effect on Immediate Data 16|32
0	None	None
1	Sign-Extend Data8 to Fill 16-Bit or 32-Bit Destination	None

10.2.3.7 Encoding of Conditional Test (tttn) Field

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with n indicating to use the condition (n = 0) or its negation (n = 1), and ttt giving the condition to test.

Mnemonic	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

10.2.3.8 Encoding of Control or Debug or Test Register (eee) Field

For the loading and storing of the Control, Debug and Test registers.

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	CR2
011	CR3
Do not use any other encoding	

When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding	

When Interpreted as Test Register Field

eee Code*	Reg Name
011	TR3
100	TR4
101	TR5
110	TR6
111	TR7
Do not use any other encoding	

	Instruction								Optional Fields	
	First Byte				Second Byte					
1	11011	OPA		1	mod		1	OPB	r/m	s-i-b	disp
2	11011	MF		OPA	mod		OPB		r/m	s-i-b	disp
3	11011	d	P	OPA	1	1	OPB		ST(i)		
4	11011	0	0	1	1	1	1	OP			
5	11011	0	1	1	1	1	1	OP			
	15–11	10	9	8	7	6	5	4 3 2 1 0			

10.2.4 ENCODING OF FLOATING POINT INSTRUCTION FIELDS

Instructions for the FPU assume one of the five forms shown in the following table. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011B.

OP = Instruction opcode, possible split into two fields OPA and OPB

MF = Memory Format
- 00—32-bit real
- 01—32-bit integer
- 10—64-bit real
- 11—16-bit integer

P = Pop
- 0—Do not pop stack
- 1—Pop stack after operation

d = Destination
- 0—Destination is ST(0)
- 1—Destination is ST(i)

R XOR d = 0—Destination (op) Source
R XOR d = 1—Source (op) Destination

ST(i) = Register stack element *i*
- 000 = Stack top
- 001 = Second stack element
- •
- •
- •
- 111 = Eighth stack element

mod (Mode field) and r/m (Register/Memory specifier) have the same interpretation as the corresponding fields of the integer instructions.

s-i-b (Scale Index Base) byte and disp (displacement) are optionally present in instructions that have mod and r/m fields. Their presence depends on the values of mod and r/m, as for integer instructions.

11.0 DIFFERENCES BETWEEN THE Intel486™ DX2 MICROPROCESSOR AND THE Intel386™ MICROPROCESSOR PLUS THE INTEL387 MATH COPROCESSOR EXTENSION

The differences between the Intel486 DX2 microprocessor and the Intel386 microprocessor are due to performance enhancements. The differences between the microprocessors are listed below.

1. Instruction clock counts have been reduced to achieve higher performance. See Section 10.
2. The Intel486 DX2 microprocessor bus is significantly faster than the Intel386 microprocessor bus. Differences include an internally doubled clock, parity support, burst cycles, cacheable cycles, cache invalidate cycles and 8-bit bus support. The Hardware Interface and Bus Operation Sections (Sections 6 and 7) of the data sheet should be carefully read to understand the Intel486 DX2 microprocessor bus functionality.
3. To support the on-chip cache new bits have been added to control register 0 (CD and NW) (Section 2.1.2.1), new pins have been added to the bus (Section 6) and new bus cycle types have been added (Section 7). The on-chip cache needs to be enabled after reset by clearing the CD and NW bit in CR0.
4. The complete Intel387 math coprocessor instruction set and register set have been added. No I/O cycles are performed during Floating Point instructions. The instruction and data pointers are set to 0 after FINIT/FSAVE. Interrupt 9 can no longer occur, interrupt 13 occurs instead.
5. The Intel486 DX2 microprocessor supports new floating point error reporting modes to guarantee DOS compatibility. These new modes required a new bit in control register 0 (NE) (Section 2.1.2.1) and new pins (FERR# and IGNNE#) (Section 6.2.13 and 7.2.14).
6. In some cases FERR# is asserted when the next floating point instruction is encountered and in other cases it is asserted before the next floating point instruction is encountered, depending upon the execution state the instruction causing exception (see Sections 6.2.13 and 7.2.14). For both of these cases, the Intel387 Math Coprocessor asserts ERROR# when the error occurs and does not wait for the next floating point instruction to be encountered.
7. Six new instructions have been added:
   Byte Swap (BSWAP)
   Exchange-and-Add (XADD)
   Compare and Exchange (CMPXCHG)
   Invalidate Data Cache (INVD)
   Write-back and Invalidate Data Cache (WBINVD)
   Invalidate TLB Entry (INVLPG)
8. There are two new bits defined in control register 3, the page table entries and page directory entries (PCD and PWT) (Section 4.5.2.5).
9. A new page protection feature has been added. This feature required a new bit in control register 0 (WP) (Section 2.1.2.1 and 4.5.3).
10. A new Alignment Check feature has been added. This feature required a new bit in the flags register (AC) (Section 2.1.1.3) and a new bit in control register 0 (AM) (Section 2.1.2.1).
11. The replacement algorithm for the translation lookaside buffer has been changed from a random algorithm to a pseudo least recently used algorithm like that used by the on-chip cache. See Section 5.5 for a description of the algorithm.
12. Three new testability registers, TR3, TR4 and TR5, have been added for testing the on-chip cache. TLB testability has been enhanced. See Section 8.
13. The prefetch queue has been increased from 16 bytes to 32 bytes. A jump always needs to execute after modifying code to guarantee correct execution of the new instruction.
14. After reset, the ID in the upper byte of the DX register is 04. The contents of the base registers including the floating point registers may be different after reset.

12.0 PENTIUM™ OVERDRIVE™ PROCESSOR SOCKET

This chapter contains the specifications for the Pentium OverDrive Processor Socket for systems based on the Intel486 DX2 Microprocessor. All of the specifications described herein are based on the specifications of the Intel486 DX2 Microprocessor.

One of the most important features of the Intel486 family architecture, compared with previous Intel architectures, is its "end user easy" upgradability via the Pentium OverDrive Processor Socket. Inclusion of the socket in systems based on the Intel486 family of microprocessors provides the end user with an easy and cost-effective way to increase system performance. The paradigm of simply installing an additional component into an empty socket to achieve enhanced system performance is familiar to the millions of end users and dealers who have purchased Intel Math CoProcessor upgrades to boost system floating point performance. The Pentium OverDrive Processor will provide up to 50% integer performance improvement and up to 150% floating point performance improvement over the base system performance. The Pentium OverDrive Processor takes advantage of Intel's Pentium processor technology to provide this performance improvement.

The Pentium OverDrive Processor will implement a superset of the Intel486 DX2 Microprocessor signals. The new signals for the socket, in addition to the Intel486 DX2 CPU signals, support a writeback protocol for the on-chip cache in Intel's future processors. Implementation of the cache writeback capability for the Pentium OverDrive Processor Socket is optional, although implementation of the Level 1 writeback protocol enables maximum performance gain. The signals required to implement this writeback are detailed in a separate document and are marked reserved in this databook. For more information, please contact Intel.

As a new system architecture feature, the provision of the Pentium OverDrive Processor Socket as a means for PC users to take advantage of the ever more rapid advances in software and hardware technology will help to maintain the competitiveness of Intel architecture PC-compatible systems over other architectures.

The majority of upgrade installations which take advantage of the Pentium OverDrive Processor Socket will be performed by end users and resellers. Therefore, it is important that the design be "end user easy", and that the amount of training and technical expertise required to install the Pentium OverDrive Processors be minimized. Upgrade installation instructions should be clearly described in the system user's manual. In addition, by making installation simple and foolproof, PC manufacturers can reduce the risk of system damage, warranty claims and service calls. Feedback from Intel's Math CoProcessor upgrade customers highlights three main characteristics of end user easy designs: accessible OverDrive Processor Socket location, clear indication of upgrade component orientation, and minimization of insertion force.

Upgrade Socket Location: The Pentium OverDrive Processor Socket can be located on either the motherboard or modular CPU card. The socket should be easily accessible for installation and readily visible when the PC case is removed. The Pentium OverDrive Processor Socket should not be located in a position that requires removal of any other hardware (such as hard disk drives) in order to install the Pentium OverDrive Processor.

Component Orientation: The most common mistake made by end users and resellers when installing Math CoProcessor upgrades is incorrect orientation of the chip. This can result in irreversible damage to the chip and/or the PC. To solve this problem, Intel has designed the Pentium OverDrive Processor Socket and the Pentium OverDrive Processor with a keying mechanism to ensure proper orientation of the upgrade component by the PC user. The keying mechanism for the Pentium OverDrive Processor is four missing pins on one corner of the device. To be effective as a keying mechanism the corresponding locations in the socket must be plugged. The Pentium OverDrive Processor Socket is designed to be backward compatible with the 169-pin OverDrive Socket of Intel486 SX and Intel486 DX systems. In order to maintain compatibility, the Pentium OverDrive Processor Socket should include the Key Pin at location E5. In addition, the location of the key corner should be clearly marked on the motherboard or CPU card, for example by silk screening.

Insertion Force: The third major concern voiced by end users refers to how much pressure should be exerted on the upgrade chip and PC board for prop-

er installation without damage. This becomes even more of a concern with the larger components which require up to 200 pounds of pressure for insertion into a standard screw machine socket. This level of pressure can easily result in cracked traces and stress to solder joints. To minimize the risk of system damage, it is recommended that a Zero Insertion Force (ZIF) socket be used for the Pentium OverDrive Processor Socket. Designing with a ZIF socket eliminates the need to design in additional structural support to prevent flexing of the PC board during installation, and results in improved end user and reseller product satisfaction due to easy "drop-in" installation.

12.0.1 Pentium™ OVERDRIVE™ PROCESSOR SOCKET OVERVIEW

The Pentium OverDrive Processor Socket is designed such that when a Pentium OverDrive Processor is installed in the socket, the original CPU relinquishes control of the system to the Pentium OverDrive Processor by backing off the bus. The circuit design requirements for the Pentium OverDrive Processor Socket are discussed in Section 12.1. In addition to the Pentium OverDrive Processor Socket circuits, there are layout considerations for the socket and processor spatial requirements. These issues are discussed in Section 12.2. Because the Pentium OverDrive Processor must function in the socket, the Pentium OverDrive Processor Socket heat dissipation specifications must be implemented. Section 12.3 shows the Pentium OverDrive Processor heat dissipation requirements for a hypothetical system design at 25 MHz and 33 MHz. Because the system must operate correctly with any OverDrive Processor without a BIOS change, BIOS and software restrictions and recommendations are provided in Section 12.4. Section 12.5 discusses Pentium OverDrive Processor Socket test requirements. Finally, Sections 12.6 and 12.7 specify the pinout and electrical characteristics of the Pentium OverDrive Processor, respectively.

12.1 Pentium™ OverDrive™ Processor Circuit Design

The Pentium OverDrive Processor Socket is designed to reside on the same processor bus as the Intel486 DX2 CPU. This socket specifies a UP# output (Upgrade Present) pin which should be connected directly to the UP# input pin of the Intel486 DX2 Microprocessor. When the Pentium OverDrive Processor occupies the socket, the UP# signal (active low) forces the Intel486 DX2 Microprocessor to 3-state all outputs and reduce power consumption. When the Pentium OverDrive Processor is not in the socket, a pullup resistor, internal to the Intel486 DX2 Microprocessor, drives UP# inactive and allows the Intel486 DX2 Microprocessor to control the processor bus.

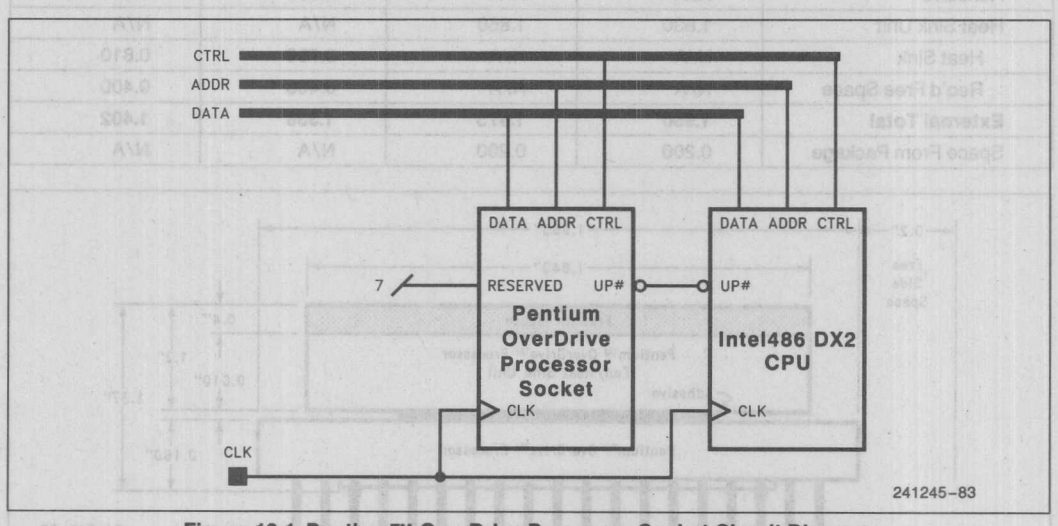


Figure 12.1. Pentium™ OverDrive Processor Socket Circuit Diagram

12.2 Socket Layout

This section discusses four aspects for the Pentium OverDrive Processor Socket: compatibility, size, upgradability, and vendors.

12.2.1 BACKWARD COMPATIBILITY

The Pentium OverDrive Processor Socket for Intel486 DX2 Microprocessor-based systems is designed to be compatible with the OverDrive Processor for Intel486 SX CPU- and Intel486 DX CPU-based systems.

The Pentium OverDrive Processor Socket has a fourth row of contacts around the outside of the 169 contacts defined for the Intel486 SX CPU- and Intel486 DX CPU-based OverDrive Processor sockets. The three inner rows, with inner key pin, are 100% compatible with the 169-pin PGA OverDrive Processor, for Intel486-based systems. For backward compatibility, the inner row key pin location (E5) must be included in the Pentium OverDrive Processor Socket.

12.2.2 MECHANICAL DESIGN CONSIDERATIONS

The Pentium OverDrive Processor is designed to fit in a standard 240-lead (19 x 19) PGA socket with four corner pins removed. The Pentium OverDrive Processor uses an active heat sink, and therefore, requires vertical clearance to allow adequate air circulation.

The maximum and minimum dimensions of the Pentium OverDrive Processor package with a fan/heat sink are shown in Table 12.1. The fan/heat sink unit is divided into the size of the actual heat sink, and the required free space above the heat sink. The total height required for the Pentium OverDrive Processor from the motherboard will depend on the height of the PGA socket. The total external height given in Table 12.1 is only measured from the PGA pin stand-offs. Table 12.1 also details the minimum clearance needed around all four sides of the PGA package.

Table 12.1. Pentium OverDrive Processor, 236-Pin, PGA Package Dimensions with Active Heat Sink Attached

Component	Length and Width (inches)		Height (inches)	
	Minimum	Maximum	Minimum	Maximum
PGA Package	1.950	1.975	0.140	0.180
Adhesive	N/A	N/A	0.008	0.012
Heat Sink Unit	1.830	1.850	N/A	N/A
Heat Sink	N/A	N/A	0.790	0.810
Req'd Free Space	N/A	N/A	0.400	0.400
External Total	**1.950**	**1.975**	**1.338**	**1.402**
Space From Package	0.200	0.200	N/A	N/A

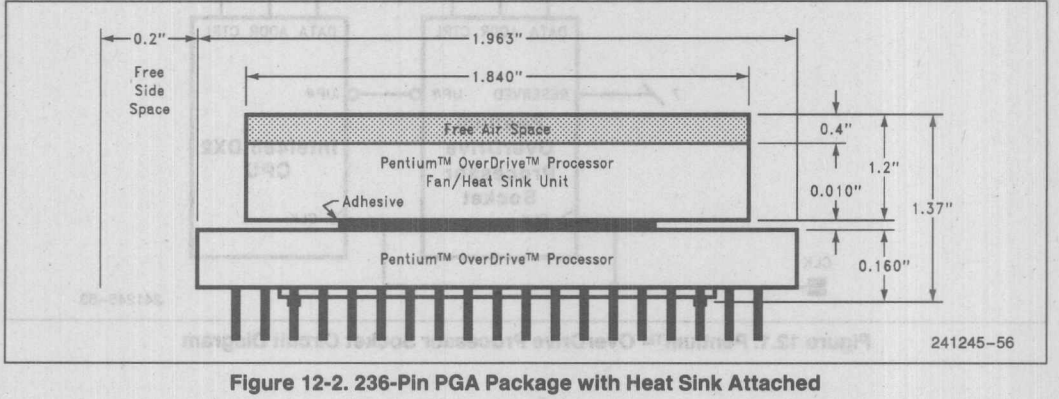


Figure 12-2. 236-Pin PGA Package with Heat Sink Attached

Since the Pentium OverDrive Processor dissipates more power than the Intel486 CPU family members, it requires a larger cooling capacity. To facilitate the task of cooling the Pentium OverDrive Processor, Intel will ship the product with a fan/heat sink. No external connections (i.e., power) will be required for the fan/heat sink. All the needed connections will be made through the pins of the processor. The amount of extra power needed for the fan is accounted for in the I_{CC} numbers of the processor (see Section 12.3). To ensure adequate air circulation, the additional clearance specified in Table 12.1 must be provided.

12.2.3 "END USER EASY" RECOMMENDATIONS

PC buyers value easy and safe upgrade installation. PC manufacturers can make upgrade component installation in the Pentium OverDrive Processor socket simple and foolproof for the end user and reseller by implementing the suggestions listed in Table 12.2.

Table 12.2. Socket and Layout Considerations

"End User Easy" Feature	Implementation
Visible Pentium OverDrive Processor Socket	The Pentium OverDrive Processor Socket should be easily visible when the PC's cover is removed. Label the Pentium OverDrive Processor Socket and the location of pin 1 by silk screening this information on the PC board.
Accessible Pentium OverDrive Processor Socket	Make the Pentium OverDrive Processor Socket easily accessible to the end user (i.e., do not place the Pentium OverDrive Processor Socket under a disk drive). Be sure to leave enough clearance to open the Zero Insertion Force (ZIF) socket.
Foolproof Chip Orientation	This Pentium OverDrive Processor Socket must insure proper orientation of not only the Pentium OverDrive Processor but also the OverDrive Processor for Intel486 SX CPU based systems. The PGA package of the Pentium OverDrive Processor is oriented by the four corner pins that have been removed from the "pin 1" corner. These four contacts (A2, A3, B1 and C1) in the socket should be plugged, such that PGA pins cannot be inserted, to assure correct orientation. The 169 pin, PGA package of the OverDrive Processor for Intel486 SX CPU systems is oriented by the "key" pin located in the inside corner of the "pin 1" corner. All inside contacts (11 innermost rows) should be plugged, except the "key" pin (E5), to insure correct orientation and alignment. The total number of contacts for the Pentium OverDrive Processor Socket is therefore 237; a standard 240-pin socket plus the inside "key" pin and less the four outside corner pins. Supplying a 237-pin socket as the Pentium OverDrive Processor Socket eliminates the possibility of end users or resellers damaging the PC board or the Pentium OverDrive Processor by powering up the system with the Pentium OverDrive Processor in an incorrect orientation.
Zero Insertion Force Pentium OverDrive Processor Socket	The high pin count of the Pentium OverDrive Processor makes the insertion force required for installation into a screw machine PGA socket excessive. Even most Low Insertion Force (LIF) sockets often require more than 60 lbs. of insertion force. A Zero Insertion Force (ZIF) socket insures that the chip insertion force does not damage the PC board. Be sure to allow enough clearance for the ZIF socket handle. Do not use a LIF or screw machine socket.
"Plug and Play"	Jumper or switch changes should not be needed to electrically configure the system for the Pentium OverDrive Processor.
Thorough Documentation	Describe the Pentium OverDrive Processor Socket and the Pentium OverDrive Processor installation procedure in the PC's User's Manual.

12.2.4 ZIF SOCKET VENDORS

The following lists provide socket vendor information based on products offered for the OverDrive Socket for Intel486 DX and Intel486 SX CPU Systems

NOTE:
This is not a comprehensive list. Intel cannot guarantee that these sockets will meet every PC manufacturer's specific requirements.

Zero Insertion Force OverDrive Processor Sockets and Vendors:

1. AMP Inc.
   219 American Avenue
   Greensboro, N.C. 27409-1803
   Part Number: TBD
   Contact: James Crompton - (919) 855-2338
2. Yamaichi Electronics
   1420 Koll Circle, Suite B
   San José, CA 95112
   Part Number: TBD
   Contact: Jim Bennett, Sales Manager - (408) 452-0797

12.3 Thermal Design Considerations

The Pentium OverDrive Processor and system chassis have several unique design requirements due to the attached active heat sink. The following sections provide sample maximum system operating temperature calculations so systems may be designed to comply with the thermal requirements of the Pentium OverDrive Processor.

Thermal Calculations for a Hypothetical System

The following equation can be used to calculate the maximum operating temperature of a system:

$$T_A(in) = T_{SINK} - (Power * \theta_{SI})$$

The parameters are defined as follows:

T_A(in): The temperature of the air going **into** the fan/heat sink.

T_{SINK}: Temperature of heat sink base, as measured in the center.

Power: Dissipation in Watts = $V_{CC} * I_{CC}$

θ_{SI}: Heat Sink to Internal Temperature [T_A(in)] Thermal Resistance

T_A (out): The temperature of the air outside the system.

Since the Pentium OverDrive Processor uses an active heat sink, θ_{SI} (as shown in Table 12.3) is relatively constant, regardless of the airflow provided to the processor. Table 12.4 details the maximum current requirements of the Pentium OverDrive Processor. The maximum ambient temperature specification for the Pentium OverDrive Processor is 55°C for both 25 MHz and 33 MHz processors with the heat sink attached. Therefore, the internal temperature of the air (T_A (in)) may not exceed 55°C under the worst case operating conditions specified for the system. This ensures that the value of T_{SINK} does not exceed 85°C.

Table 12.3. Thermal Resistance (°C/W)—θ_{SI}

Processor Type	θ_{SI} – °C/W
Fan/Heat Sink	2.4

Table 12.4. OverDrive Processor Typical and Maximum I_{CC} Values

System Frequency (MHz)	Processor Typical I_{CC} (mA)	Processor Maximum I_{CC} (mA)
25	TBD	1900
33	TBD	2500

I_{CC} is dependent upon the V_{CC} level of the system, processor bus loading, software code sequences, and silicon process variations.

Maximum T_A(in) is specified and be verified using the equation and parameters provided

$$T_A(in) = T_{SINK} - (Power * \theta_{SI})$$

$$T_A(in) = 85°C - ((2.5A * 5V) * 2.4°C/W)$$

$$T_A(in) = 85°C - ((12.5W * 2.4°C/W)$$

$$T_A(in) = 85°C - 30°C$$

$$T_A(in) = 55°C$$

Assuming the internal system ambient T_A(in) is within 5°C–10°C of T_A(out), this would allow the maximum T_A(out) temperature to be approximately 45°C–50°C. It is the responsibility of the system designer to ensure T_A(in) meets this specification by providing sufficient airflow around the Pentium OverDrive Processor to remove the heated air expelled by the fan/heat sink.

12.4 BIOS and Software

The following should be considered when designing the Pentium OverDrive Processor Socket for an Intel486 DX2 microprocessor-based system.

12.4.1 OverDrive PROCESSOR DETECTION

The component identifier and stepping/revision identifier for the Pentium OverDrive Processor is readable in the DH and DL registers respectively, immediately after RESET, where

DH = 15h

DL = 30h–3Fh

As with the Intel486 DX2 microprocessor specification, it is recommended that the BIOS save the contents of the DX register, immediately after RESET, so that this information can be used later, if required.

12.4.2. TIMING DEPENDENT LOOPS

The Pentium OverDrive Processor for Intel486 DX2 microprocessor-based systems executes instructions at a multiple of the frequency of the input clock. This Pentium OverDrive Processor also will use advanced design techniques to decrease the number of clocks per instruction (cpi) from that of the Intel486 DX2 microprocessor. Thus software, such as instruction-based timing loops, will execute faster on the Pentium OverDrive Processor than on either the Intel486 DX CPU or the Intel486 DX2 microprocessor at the same input clock frequency. Instructions such as NOP, LOOP, and JMP $+2 are frequently used by the BIOS to implement timing loops that are required, for example, to enforce recovery time between consecutive accesses for I/O devices. These instruction-based, timing-loop implementations may require modification to be compatible with this Pentium OverDrive Processor Socket.

In order to avoid any incompatibilities, timing loops can be implemented in hardware rather than in software. This provides transparency and also does not require any change in BIOS or I/O device drivers in the future when moving to higher processor clock speeds.

As an example, a timing loop may be implemented as follows: The software performs a dummy I/O instruction to an unused I/O port. The hardware for the bus controller logic recognizes this I/O instruction and delays the termination of the I/O cycle by keeping RDY# or BRDY# deasserted for the appropriate amount of time.

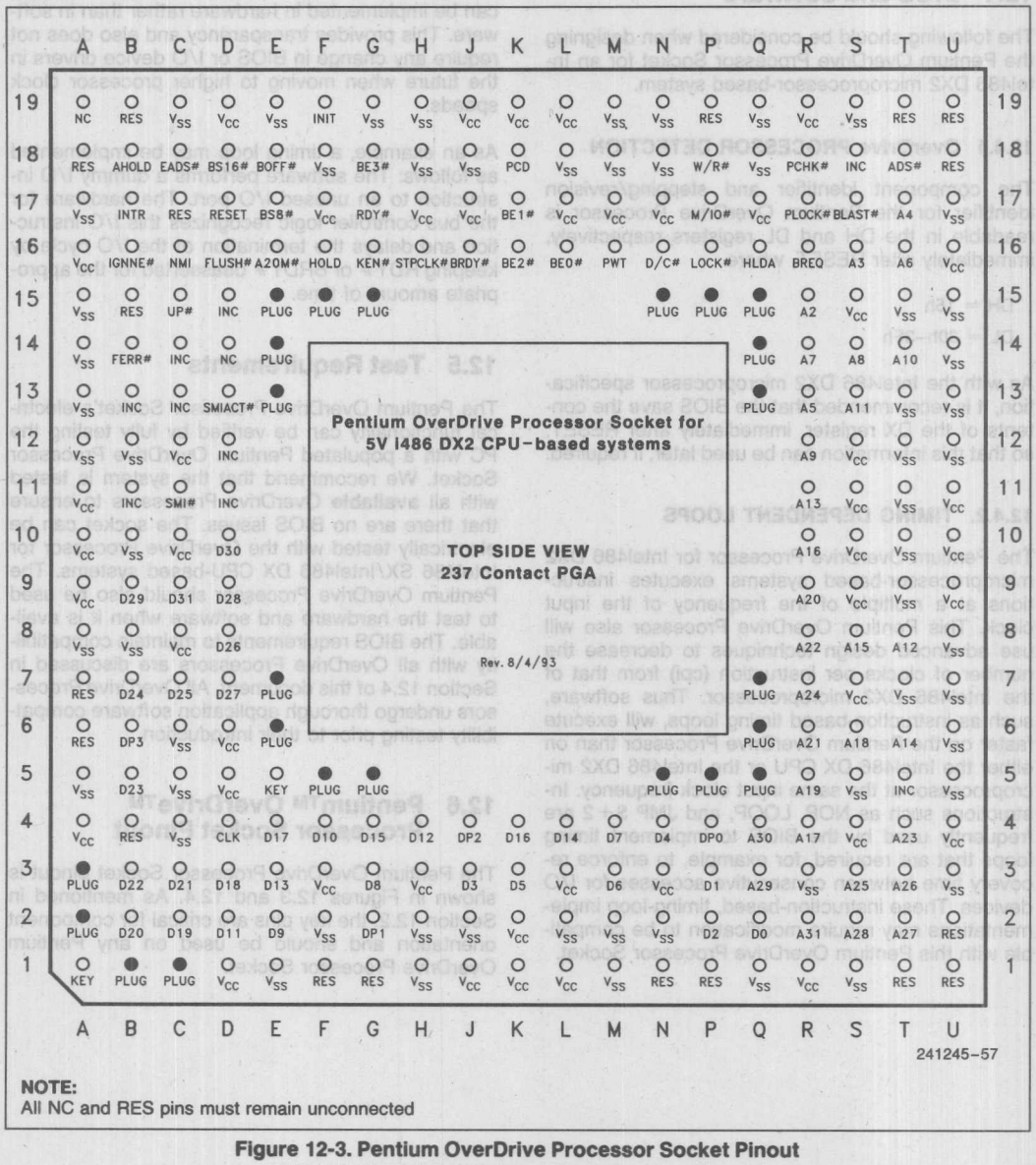
12.5 Test Requirements

The Pentium OverDrive Processor Socket's electrical functionality can be verified by fully testing the PC with a populated Pentium OverDrive Processor Socket. We recommend that the system is tested with all **available** OverDrive Processors to ensure that there are no BIOS issues. The socket can be electrically tested with the OverDrive Processor for Intel486 SX/Intel486 DX CPU-based systems. The Pentium OverDrive Processor should also be used to test the hardware and software when it is available. The BIOS requirements to maintain compatibility with all OverDrive Processors are discussed in Section 12.4 of this document. All OverDrive Processors undergo thorough application software compatibility testing prior to their introduction.

12.6 Pentium™ OverDrive™ Processor Socket Pinout

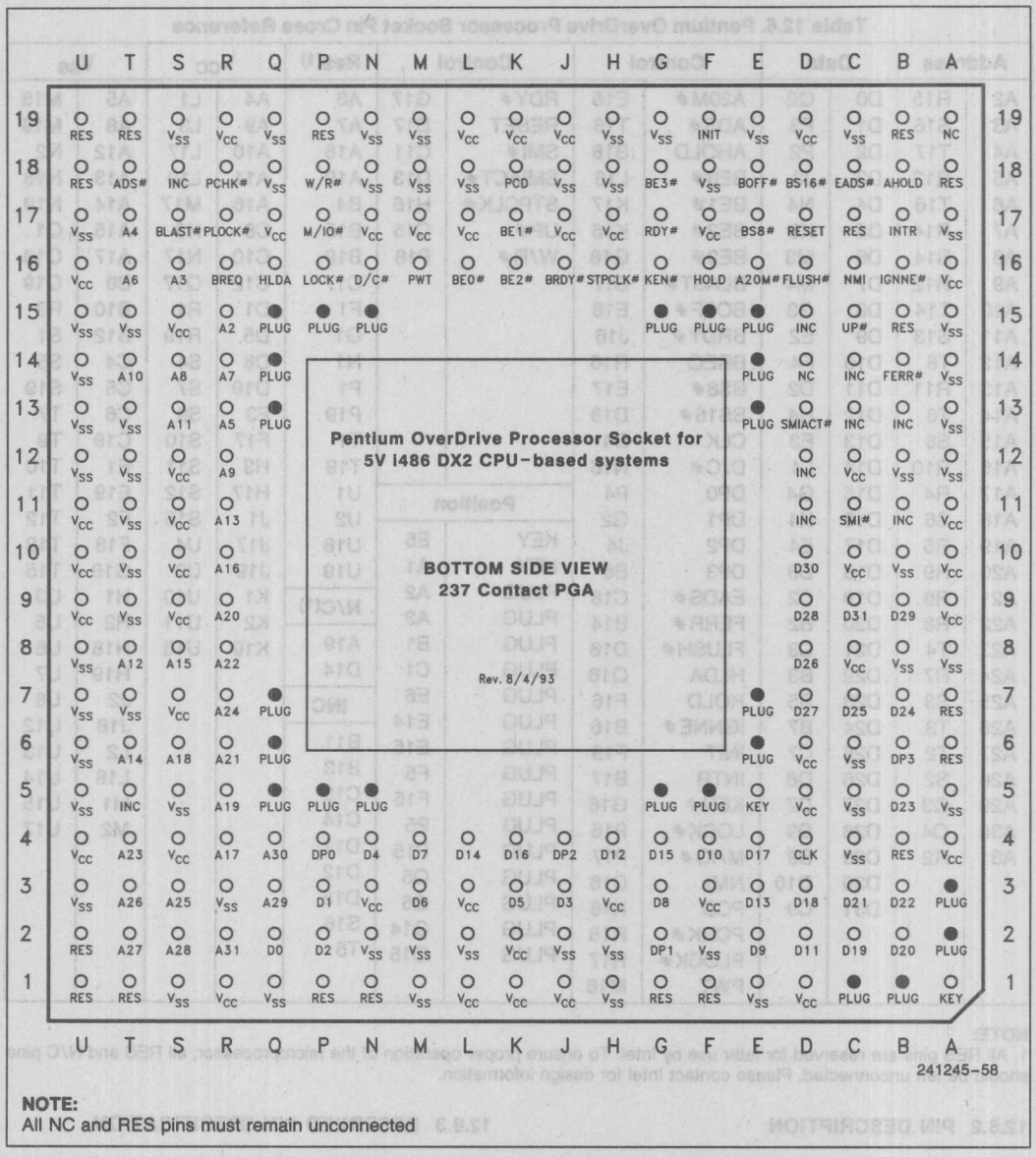
The Pentium OverDrive Processor Socket pinout is shown in Figures 12.3 and 12.4. As mentioned in Section 12.2, the key pins are critical for component orientation and should be used on any Pentium OverDrive Processor Socket.

12.6.1 PINOUT



NOTE:
All NC and RES pins must remain unconnected

Figure 12-3. Pentium OverDrive Processor Socket Pinout for 5V Intel486 DX2 CPU-Based Systems (Top Side View)



NOTE:
All NC and RES pins must remain unconnected

Figure 12-4. Pentium OverDrive Processor Socket Pinout for 5V Intel486 DX2 CPU-Based System (Bottom Side View)

Table 12.6. Pentium OverDrive Processor Socket Pin Cross Reference

Address		Data		Control		Control		Res(1)	Vcc		Vss	
A2	R15	D0	Q2	A20M#	E16	RDY#	G17	A6	A4	L1	A5	M18
A3	S16	D1	P3	ADS#	T18	RESET	D17	A7	A9	L3	A8	M19
A4	T17	D2	P2	AHOLD	B18	SMI#	C11	A18	A10	L17	A12	N2
A5	R13	D3	J3	BE0#	L16	SMIACT#	D13	A19	A11	L19	A13	N18
A6	T16	D4	N4	BE1#	K17	STPCLK#	H16	B4	A16	M17	A14	N19
A7	R14	D5	K3	BE2#	K16	UP#	C15	B15	C8	N3	A15	Q1
A8	S14	D6	M3	BE3#	G18	W/R#	P18	B19	C10	N17	A17	Q18
A9	R12	D7	M4	BLAST#	S17			C17	C12	Q17	B8	Q19
A10	T14	D8	G3	BOFF#	E18			F1	D1	R1	B10	R3
A11	S13	D9	E2	BRDY#	J16			G1	D5	R19	B12	S1
A12	T8	D10	F4	BREQ	R16			N1	D6	S4	C4	S5
A13	R11	D11	D2	BS8#	E17			P1	D19	S7	C5	S19
A14	T6	D12	H4	BS16#	D18			P19	F3	S9	C6	T7
A15	S8	D13	E3	CLK	D4			T1	F17	S10	C19	T9
A16	R10	D14	L4	D/C#	N16			T19	H3	S11	E1	T10
A17	R4	D15	G4	DP0	P4	**Position**		U1	H17	S12	E19	T11
A18	S6	D16	K4	DP1	G2			U2	J1	S15	F2	T12
A19	R5	D17	E4	DP2	J4	KEY	E5	U18	J17	U4	F18	T13
A20	R9	D18	D3	DP3	B6	KEY	A1	U19	J19	U9	G19	T15
A21	R6	D19	C2	EADS#	C18	PLUG	A2	**N/C(1)**	K1	U10	H1	U3
A22	R8	D20	B2	FERR#	B14	PLUG	A3		K2	U11	H2	U5
A23	T4	D21	C3	FLUSH#	D16	PLUG	B1	A19	K19	U16	H18	U6
A24	R7	D22	B3	HLDA	Q16	PLUG	C1	D14			H19	U7
A25	S3	D23	B5	HOLD	F16	PLUG	E6	**INC**			J2	U8
A26	T3	D24	B7	IGNNE#	B16	PLUG	E14				J18	U12
A27	T2	D25	C7	INIT	F19	PLUG	E15	B11			L2	U13
A28	S2	D26	D8	INTR	B17	PLUG	F5	B13			L18	U14
A29	Q3	D27	D7	KEN#	G16	PLUG	F15	C13			M1	U15
A30	Q4	D28	D9	LOCK#	P16	PLUG	P5	C14			M2	U17
A31	R2	D29	B9	M/IO#	P17	PLUG	P15	D11				
		D30	D10	NMI	C16	PLUG	Q5	D12				
		D31	C9	PCD	K18	PLUG	Q6	D15				
				PCHK#	R18	PLUG	Q14	S18				
				PLOCK#	R17	PLUG	Q15	T5				
				PWT	M16							

NOTE:
1. All RES pins are reserved for later use by Intel. To ensure proper operation of the microprocessor, all RES and N/C pins should be left unconnected. Please contact Intel for design information.

12.6.2 PIN DESCRIPTION

The signal pin descriptions for the Pentium OverDrive Processor are identical to the pin descriptions for the Intel486 DX2 Microprocessor except for the Upgrade Present pin (UP#) and KEY pin. The pin descriptions for these two signals are shown in Table 12.7.

12.6.3 RESERVED PIN SPECIFICATION

Many pins in the Pentium OverDrive Processor Socket are defined as reserved (RES). The function of these pins is documented separately. These signals will be used to implement a Write Back level 1 (on-chip) cache protocol. These pins must not be connected unless they are used to implement a level 1 Write Back solution using the information available separately. To insure proper operation, pins marked as NC must be left unconnected as well.

Table 12.7. Pentium OverDrive Processor Socket Pin Description

Symbol	Type	Name and Function
Intel486 DX2 CPU INTERFACE		
UP#	O	The *Upgrade Present* pin is used to signal the Intel486 DX2 microprocessor to float its outputs and stop driving the bus. It is active low and is never floated. UP# is driven low at power-up and remains active for the entire duration of the Pentium OverDrive Processor operation.
KEY PIN		
KEY		The Key pin is an electrically non-functional pin which provides backward compatibility to the OverDrive Processor for Intel486 SX/Intel486 DX CPU-based systems and is used to ensure correct orientation for 169-pin upgrade products. Proper orientation of the Pentium OverDrive Processor is insured by the four socket contacts which must be plugged (A2, A3, B1 and C1); the Pentium OverDrive Processor will not have pins in these locations.

12.7 D.C./A.C. Specifications

The electrical specifications in this section represent the electrical interface of the Pentium OverDrive Processor. The Pentium OverDrive Processor will be compatible to the maximum ratings and A.C. Specifications of the Intel486 DX2 Microprocessor. Table 12.8 provides the D.C. Operating Conditions for the Pentium OverDrive Processor.

Table 12.8. Pentium OverDrive Processor Socket D.C. Parametric Values(1)

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 2)
V_{OH}	Output High Voltage	2.4		V	(Note 3)
I_{CC}	Power Supply Current CLK = 25 MHz CLK = 33 MHz		 1900 2500	 mA	
I_{LI}	Input Leakage Current		±15	μA	(Note 4)
I_{IH}	Input Leakage Current		200	μA	(Note 5)
I_{IL}	Input Leakage Current		−400	μA	(Note 6)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		13	pF	F_C = 1 MHz(7)
C_O	I/O or Output Capacitance		17	pF	F_C = 1 MHz(7)
C_{CLK}	CLK Capacitance		15	pF	F_C = 1 MHz(7)

NOTES:
1. Functional operating range: V_{CC} = 5V; T_S = 0°C to +80°C.
2. This parameter is measured at:
   — Address, Data, BEn 4.0 mA
   — Definition, Control 5.0 mA
3. This parameter is measured at:
   — Address, Data, BEn −1.0 mA
   — Definition, Control −0.9 mA
4. This parameter is for inputs without pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with pulldowns and V_{IH} = 2.4V.
6. This parameter is for inputs with pullups and V_{IL} = 0.45V.
7. Not 100% tested.

13.0 CONVERTING AN EXISTING Intel486™ DX CPU DESIGN

Converting an Intel486 DX CPU system design to an Intel486 DX2 CPU design provides more performance for a small difference in cost. Three conversion possibilities are available as shown in Table 13.1. Migrating from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU could increase performance by 35%, and migrating from a 25 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU could increase performance by an average of 70%. See the *Intel486™ DX2 Microprocessor Performance Brief* (Order #241254) for more details on performance. Conversion can be as easy as replacing one or two devices.

Table 13.1. Converting Intel486™ DX CPU Designs to Intel486™ DX2 CPU Designs

Initial Design (Intel486 DX CPU)	Converted Design (Intel486 DX2 CPU)	Typical Performance Gain
25 MHz	50 MHz	70%
33 MHz	50 MHz	35%
33 MHz	66 MHz	70%

A few system details should be checked first to be sure the design is ready for the Intel486 DX2 CPU. Check with your BIOS vendor to be sure any BIOS issues have been resolved. The BIOS for the Intel486 DX CPU may have timing loops. Since the Intel486 DX2 CPU runs instructions twice as fast as the Intel486 DX CPU, timing loops may no longer return the required results. Most of the timing loops have been removed from a standard BIOS, but there may be some versions that need updating. Another BIOS issue that may not be critical, is the processor identification code. There are different ID codes in the Intel486 DX CPU and the Intel486 DX2 CPU. The BIOS may need to be modified to identify the Intel486 DX2 CPU properly. Refer to Table 6.3 for the component ID code.

Other system parameters to watch out for are the thermal and power supply specifications. Table 14.2 details the Power Supply Current information, and Table 15.2 outlines the Thermal Resistance. Since the processor core runs twice as fast for the same input clock, the Intel486 DX2 CPU uses more power and generates more heat than the Intel486 DX CPU. Be sure that there is adequate cooling and adequate power built into the design. A heat sink is a recommended method to help provide cooling for the Intel486 DX2 CPU.

The system checks mentioned above are common to all conversions from an Intel486 DX CPU to an Intel486 DX2 CPU regardless of the speed of the processor or system.

A few system implications exist for converting from one frequency to another as shown in Figure 13.1. The first case is migrating from a 25 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU (the bus runs at the same speed for both parts). System hardware modifications need not be made to plug in the 50 MHz Intel486 DX2 CPU and achieve the desired performance. When all instructions are running out of the on-chip cache, performance increases by a maximum of 100%.

The second case is migrating from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU. This conversion is a two step process. The first step is to change the frequency source for the CPU from 33 MHz to 25 MHz. The Intel486 DX2 CPU can then be inserted into the system. Without any tuning of the memory and depending on the application, only a modest performance improvement may be observed. For programs running entirely out of the on-chip cache, however, performance can increase up to 50%. There are many factors which contribute to the performance of an application, including whether there is a second-level (L2) cache, the cache size if present, the memory subsystem design, and many other factors beyond the scope of this introduction. A comprehensive memory subsystem design guide, *AP469: Cache and Memory Design Considerations for Intel486™ DX2 Microprocessor,* is available which includes more detailed information on how each of these many factors affects Intel486 DX2 CPU-based system performance.

Because the Intel486 DX2 core runs twice as fast as its external bus, it is more sensitive to wait states. The Intel486 DX2 CPU needs to be fed instructions and data quickly. Either a high performance memory subsystem is needed or an external cache should be added. An external cache benefits the Intel486 DX2 CPU even more than it benefits the Intel486 DX CPU, and helps to hide the effects of a slower memory subsystem. The Intel486 DX CPU gains an average of 3%–9% performance by the addition of a second-level cache, but the Intel486 DX2 CPU gains an average of 20-30% performance by adding a second level cache. It should be noted however, that an external cache does not preclude the benefits of tuning the memory subsystem.

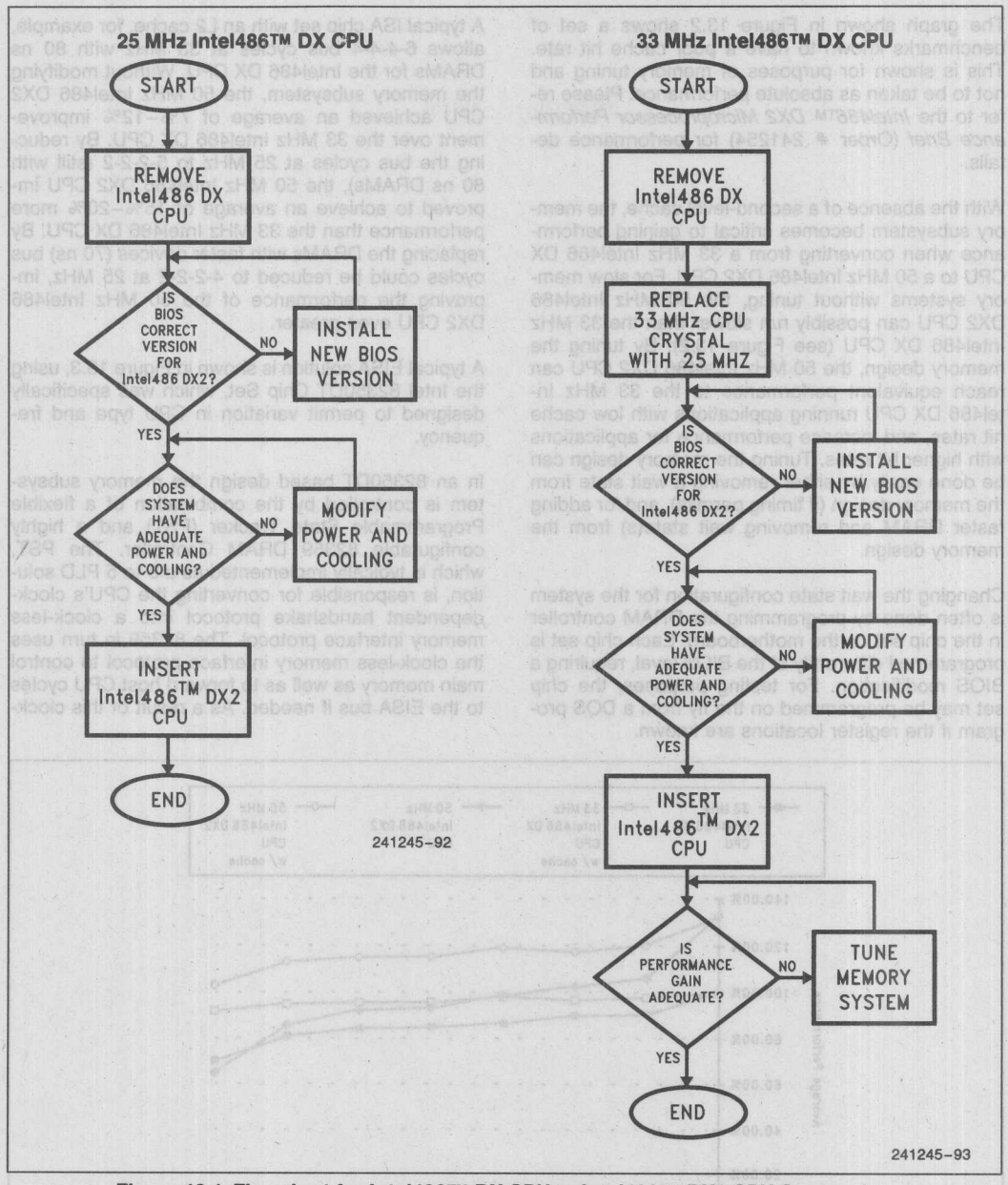


Figure 13.1. Flowchart for Intel486™ DX CPU to Intel486™ DX2 CPU Conversion

The graph shown in Figure 13.2 shows a set of benchmarks known to have a poor cache hit rate. This is shown for purposes of memory tuning and not to be taken as absolute performance. Please refer to the *Intel486™ DX2 Microprocessor Performance Brief* (Order # 241254) for performance details.

With the absence of a second-level cache, the memory subsystem becomes critical to gaining performance when converting from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU. For slow memory systems without tuning, the 50 MHz Intel486 DX2 CPU can possibly run slower than the 33 MHz Intel486 DX CPU (see Figure 13.2). By tuning the memory design, the 50 MHz Intel486 DX2 CPU can reach equivalent performance to the 33 MHz Intel486 DX CPU running applications with low cache hit rates, and increase performance for applications with higher hit rates. Tuning the memory design can be done easily by either removing a wait state from the memory design (if timing permits), and/or adding faster DRAM and removing wait state(s) from the memory design.

Changing the wait state configuration for the system is often done by programming the DRAM controller in the chip set on the motherboard. Each chip set is programmed differently at the BIOS level, requiring a BIOS modification. For testing purposes, the chip set may be programmed on the fly from a DOS program if the register locations are known.

A typical ISA chip set with an L2 cache, for example, allows 6-4-4-4 bus cycles at 33 MHz with 80 ns DRAMs for the Intel486 DX CPU. Without modifying the memory subsystem, the 50 MHz Intel486 DX2 CPU achieved an average of 7%–12% improvement over the 33 MHz Intel486 DX CPU. By reducing the bus cycles at 25 MHz to 5-2-2-2 (still with 80 ns DRAMs), the 50 MHz Intel486 DX2 CPU improved to achieve an average of 15%–20% more performance than the 33 MHz Intel486 DX CPU. By replacing the DRAMs with faster devices (70 ns) bus cycles could be reduced to 4-2-2-2 at 25 MHz, improving the performance of the 50 MHz Intel486 DX2 CPU even greater.

A typical EISA solution is shown in Figure 13.3, using the Intel 82350DT Chip Set, which was specifically designed to permit variation in CPU type and frequency.

In an 82350DT based design the memory subsystem is controlled by the combination of a flexible Programmable State Tracker (PST) and a highly configurable 82359 DRAM Controller. The PST, which is typically implemented as a 3 to 5 PLD solution, is responsible for converting the CPU's clock-dependent handshake protocol into a clock-less memory interface protocol. The 82359 in turn uses the clock-less memory interface protocol to control main memory as well as to forward host CPU cycles to the EISA bus if needed. As a result of this clock-

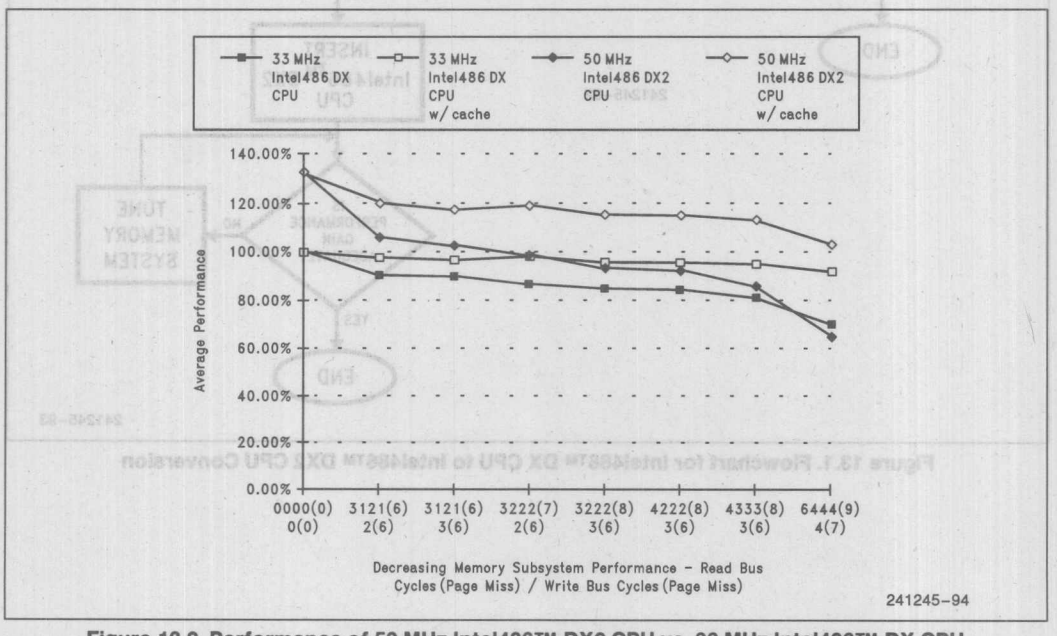


Figure 13.2. Performance of 50 MHz Intel486™ DX2 CPU vs. 33 MHz Intel486™ DX CPU

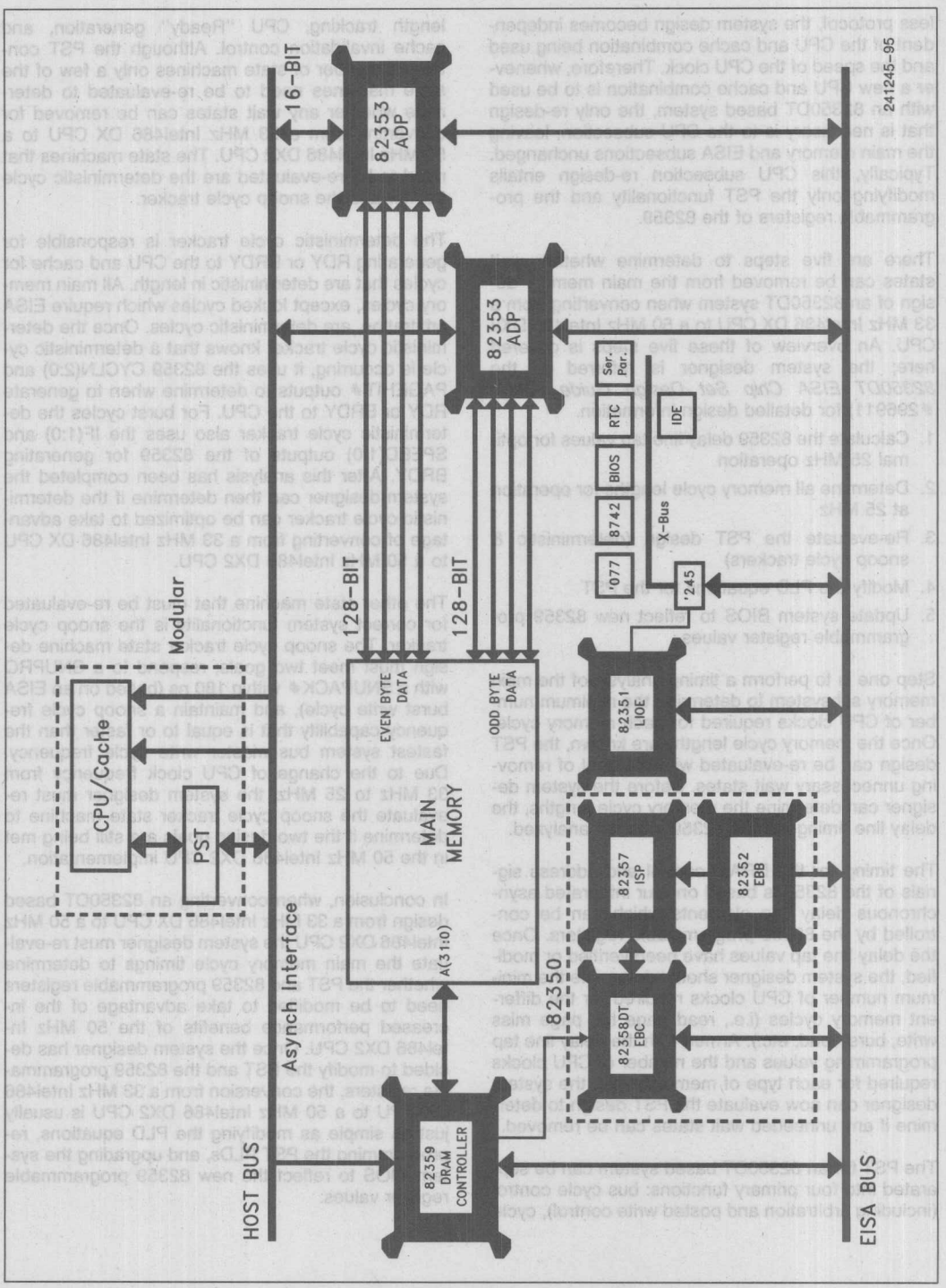


Figure 13.3. Typical 82350DT System Architecture

less protocol, the system design becomes independent of the CPU and cache combination being used and the speed of the CPU clock. Therefore, whenever a new CPU and cache combination is to be used with an 82350DT based system, the only re-design that is necessary is to the CPU subsection, leaving the main memory and EISA subsections unchanged. Typically, this CPU subsection re-design entails modifying only the PST functionality and the programmable registers of the 82359.

There are five steps to determine whether wait states can be removed from the main memory design of an 82350DT system when converting from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU. An overview of these five steps is covered here; the system designer is referred to the *82350DT EISA Chip Set Design Guide* (Order #296911) for detailed design information.

1. Calculate the 82359 delay line tap values for optimal 25 MHz operation
2. Determine all memory cycle lengths for operation at 25 MHz
3. Re-evaluate the PST design (deterministic & snoop cycle trackers)
4. Modify the PLD equations for the PST
5. Update system BIOS to reflect new 82359 programmable register values

Step one is to perform a timing analysis of the main memory subsystem to determine the minimum number of CPU clocks required for each memory cycle. Once the memory cycle lengths are known, the PST design can be re-evaluated with the goal of removing unnecessary wait states. Before the system designer can determine the memory cycle lengths, the delay line timings of the 82359 must be analyzed.

The timing for the DRAM control and address signals of the 82359 is based on four integrated asynchronous delay line elements which can be controlled by the 82359 programmable registers. Once the delay line tap values have been verified or modified, the system designer should determine the minimum number of CPU clocks required for the different memory cycles (i.e., read page hit, page miss write, burst read, etc.). Armed with the delay line tap programming values and the number of CPU clocks required for each type of memory cycle, the system designer can now evaluate the PST design to determine if any unneeded wait states can be removed.

The PST for an 82350DT based system can be separated into four primary functions: bus cycle control (including arbitration and posted write control), cycle length tracking, CPU "Ready" generation, and cache invalidation control. Although the PST contains a number of state machines only a few of the state machines need to be re-evaluated to determine whether any wait states can be removed for converting from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU. The state machines that need to be re-evaluated are the deterministic cycle tracker and the snoop cycle tracker.

The deterministic cycle tracker is responsible for generating RDY or BRDY to the CPU and cache for cycles that are deterministic in length. All main memory cycles, except locked cycles which require EISA arbitration, are deterministic cycles. Once the deterministic cycle tracker knows that a deterministic cycle is occurring, it uses the 82359 CYCLN(2:0) and PAGEHIT# outputs to determine when to generate RDY or BRDY to the CPU. For burst cycles the deterministic cycle tracker also uses the IF(1:0) and SPEED(1:0) outputs of the 82359 for generating BRDY. After this analysis has been completed the system designer can then determine if the deterministic cycle tracker can be optimized to take advantage of converting from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU.

The other state machine that must be re-evaluated for correct system functionality is the snoop cycle tracker. The snoop cycle tracker state machine design must meet two goals; respond to a SNUPRQ with a SNUPACK# within 180 ns (based on an EISA burst write cycle), and maintain a snoop cycle frequency capability that is equal to or faster than the fastest system bus master write cycle frequency. Due to the change of CPU clock frequency from 33 MHz to 25 MHz, the system designer must re-evaluate the snoop cycle tracker state machine to determine if the two design goals are still being met in the 50 MHz Intel486 DX2 CPU implementation.

In conclusion, when converting an 82350DT based design from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU the system designer must re-evaluate the main memory cycle timings to determine whether the PST and 82359 programmable registers need to be modified to take advantage of the increased performance benefits of the 50 MHz Intel486 DX2 CPU. Once the system designer has decided to modify the PST and the 82359 programmable registers, the conversion from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU is usually just as simple as modifying the PLD equations, reprogramming the PST PLDs, and upgrading the system BIOS to reflect the new 82359 programmable register values.

The following sections describe recommended electrical connections for the Intel486 DX2 microprocessor, and its electrical specifications.

14.1 Power and Grounding

14.1.1 POWER CONNECTIONS

The Intel486 DX2 microprocessor is implemented in CHMOS V technology and has modest power requirements. However, its high clock frequency output buffers can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 24 V_{CC} and 28 V_{SS} pins feed the Intel486 DX2 microprocessor.

Power and ground connections must be made to all external V_{CC} and GND pins of the Intel486 DX2 microprocessor. On the circuit board, all V_{CC} pins must be connected on a V_{CC} plane. All V_{SS} pins must be likewise connected on a GND plane.

14.1.2 POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitance should be placed near the Intel486 DX2 microprocessor. The Intel486 DX2 microprocessor driving its 32-bit parallel address and data busses at high frequencies can cause transient power surges, particularly when driving large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the Intel486 DX2 microprocessor and decoupling capacitors as much as possible. Capacitors specifically for PGA packages are also commercially available.

14.1.3 OTHER CONNECTION RECOMMENDATIONS

N.C. pins should always remain unconnected.

For reliable operation, always connect unused inputs to an appropriate signal level. Active LOW inputs should be connected to V_{CC} through a pullup resistor. Pullups in the range of 20 KΩ are recommended. Active HIGH inputs should be connected to GND.

14.2 Maximum Ratings

Table 14.1 is a stress rating only, and functional operation at the maximums is not guaranteed. Function operating conditions are given in 14.3 D.C. Specifications and 14.4 A.C. Specifications.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the Intel486 DX2 microprocessor contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

Table 14.1. Absolute Maximum Ratings

Case Temperature under Bias	−65°C to +110°C
Storage Temperature	−65°C to +150°C
Voltage on Any Pin with Respect to Ground	−0.5 to V_{CC} + 0.5V
Supply Voltage with Respect to V_{SS}	−0.5V to +6.5V

14.3 Intel486 DX2 D.C. Specifications

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

Table 14-2. DC Parametric Values

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current (66 MHz) (50 MHz)		 1200 950	 mA	 (Note 3)
I_{CCF}	Power Supply Current in Power Down Mode		50	mA	(Note 8)
I_{LI}	Input Leakage Current		±15	μA	(Note 4)
I_{IH}	Input Leakage Current		200	μA	(Note 5)
I_{IL}	Input Leakage Current		−400	μA	(Note 6)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		13	pF	F_C = 1 MHz (Note 7)
C_O	I/O or Output Capacitance		17	pF	F_C = 1 MHz (Note 7)
C_{CLK}	CLK Capacitance		15	pF	F_C = 1 MHz (Note 7)

PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
1. This parameter is measured at:
   Address, Data, BEn 4.0 mA
   Definition, Control 5.0 mA
2. This parameter is measured at:
   Address, Data, BEn −1.0 mA
   Definition, Control −0.9 mA
3. Typical supply current:
   775 mA @ 50 MHz
   975 mA @ 66 MHz
4. This parameter is for inputs without internal pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with internal pulldowns and V_{IH} = 2.4V.
6. This parameter is for inputs with internal pullups and V_{IL} = 0.45V.
7. Not 100% tested.
8. The I_{CCF} specification in the above table is a target value. It has not been tested.

14.4 A.C. Specifications

The A.C. specifications, given in Table 14.3, consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the rising edge of the CLK signal.

A.C. specifications measurement is defined by Figures 14.1–14.7. All timings are referenced to 1.5V unless otherwise specified. Inputs must be driven to the voltage levels indicated by Figure 14.3 when A.C. specifications are measured. Intel486 DX2 microprocessor output delays are specified with minimum and maximum limits, measured as shown. The minimum Intel486 DX2 microprocessor delay times are hold times provided to external circuitry. Intel486 DX2 microprocessor input setup and hold times are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct Intel486 DX2 microprocessor operation.

Table 14.4.1. 50 MHz Intel486 DX2 Microprocessor A.C. Characteristics

V_{CC} = 5V ±5%; T_{case} = 0°C to +85°C; C_l = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	25	MHz		1X Clock Driven to Intel486 DX2
t_1	CLK Period	40	125	ns	14.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	14		ns	14.1	at 2V
t_3	CLK Low Time	14		ns	14.1	at 0.8V
t_4	CLK Fall Time		4	ns	14.1	2V to 0.8V
t_5	CLK Rise Time		4	ns	14.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	19	ns	14.5	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		28	ns	14.6	(Note 1)
t_8	PCHK# Valid Delay	3	24	ns	14.4	(Note 3)
t_{8a}	BLAST#, PLOCK# Valid Delay	3	24	ns	14.5	(Note 3)
t_9	BLAST#, PLOCK# Float Delay		28	ns	14.6	(Note 1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	20	ns	14.5	(Note 3)
t_{11}	D0–D31, DP0–3 Write Data Float Delay		28	ns	14.6	(Note 1)
t_{12}	EADS# Setup Time	8		ns	14.2	
t_{13}	EADS# Hold Time	3		ns	14.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	8		ns	14.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	14.2	
t_{16}	RDY#, BRDY# Setup Time	8		ns	14.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	14.3	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	8		ns	14.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	14.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	8		ns	14.2	(Note 4)
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	14.2	(Note 4)
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	5		ns	14.2, 14.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	14.2, 14.3	

NOTES:
1. Not 100% tested. Guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF. Charts 14.4.3 provides the charts that may be used to determine the delay due to derating, depending on the lumped capacitive loading, that must be added to these specification values.
3. The minimum Intel486 DX2 output valid delays are hold times provided to external circuitry.
4. A reset pulse width of 15 CLK cycles is required for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.

Table 14.4.2 66 MHz Intel486 DX2 Microprocessor A.C. Characteristics

V_{CC} = 5V ±5%; T_{case} = 0°C to +85°C; C_l = 50 pF unless otherwise specified (Note 2)

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	33	MHz		1X Clock Driven to Intel486 DX2
t_1	CLK Period	30	125	ns	14.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	11		ns	14.1	at 2V
t_3	CLK Low Time	11		ns	14.1	at 0.8V
t_4	CLK Fall Time		3	ns	14.1	2V to 0.8V
t_5	CLK Rise Time		3	ns	14.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	14	ns	14.5	(Note 3)
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		20	ns	14.6	(Note 1)
t_8	PCHK# Valid Delay	3	14	ns	14.4	(Note 3)
t_{8a}	BLAST#, PLOCK# Valid Delay	3	14	ns	14.5	(Note 3)
t_9	BLAST#, PLOCK# Float Delay		20	ns	14.6	(Note 1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	14	ns	14.5	(Note 3)
t_{11}	D0–D31, DP0–3 Write Data Float Delay		20	ns	14.6	(Note 1)
t_{12}	EADS# Setup Time	5		ns	14.2	
t_{13}	EADS# Hold Time	3		ns	14.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	5		ns	14.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	14.2	
t_{16}	RDY#, BRDY# Setup Time	5		ns	14.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	14.3	
t_{18}	HOLD, AHOLD, Setup Time	6		ns	14.2	
t_{18a}	BOFF# Setup Time	7		ns	14.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	14.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	5		ns	14.2	(Note 4)
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	14.2	(Note 4)
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	5		ns	14.2, 14.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	14.2, 14.3	

PRELIMINARY
INTEL CONFIDENTIAL
SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
1. Not 100% tested. Guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF. Charts 14.4.3 provides the charts that may be used to determine the delay due to derating, depending on the lumped capacitive loading, that must be added to these specification values.
3. The minimum Intel486 DX2 output valid delays are hold times provided to external circuitry.
4. A reset pulse width of 15 CLK cycles is required for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.

Table 14.4.3. Intel486 DX2 Microprocessor A.C. Characteristics for Boundary Scan Test Signals

V_{CC} = 5V ±5%, T_{case} = 0°C to +85°C, C_L = 0 pF
All Inputs and Outputs are TTL Level (Note 4)

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{24}	TCK Frequency		25	MHz		1x Clock
t_{25}	TCK Period	40		ns		(Note 2)
t_{26}	TCK High Time	10		ns		at 2.0V
t_{27}	TCK Low Time	10		ns		at 0.8V
t_{28}	TCK Rise Time		4	ns		(Note 1)
t_{29}	TCK Fall Time		4	ns		(Note 1)
t_{30}	TDI, TMS Setup Time	8		ns	14.7	(Note 3)
t_{31}	TDI, TMS Hold Time	7		ns	14.7	(Note 3)
t_{32}	TDO Valid Delay	3	25	ns	14.7	(Note 3)
t_{33}	TDO Float Delay		TBD			
t_{34}	All Outputs (Non-Test) Valid Delay	3	25	ns	14.7	(Note 3)
t_{35}	All Outputs (Non-Test) Float Delay		36	ns	14.7	(Note 3)
t_{36}	All Inputs (Non-Test) Setup Time	8		ns	14.7	(Note 3)
t_{37}	All Inputs (Non-Test) Hold Time	7		ns	14.7	(Note 3)

NOTES:
1. Rise/Fall times are measured between 0.8V and 2.0V. Rise/Fall times can be relaxed by 1 ns per 10 ns increase in TCK period.
2. TCK period ≥ CLK period.
3. Parameter measured from TCK.
4. Boundary Scan A.C. Specifications in the above table are target values. They have not been characterized. Therefore they are subject to change.

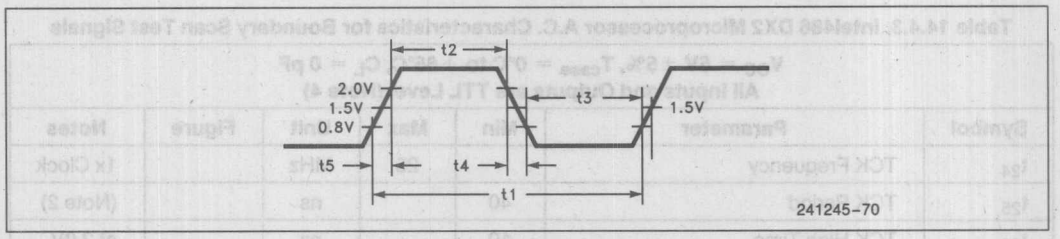


Figure 14.1. CLK Waveforms

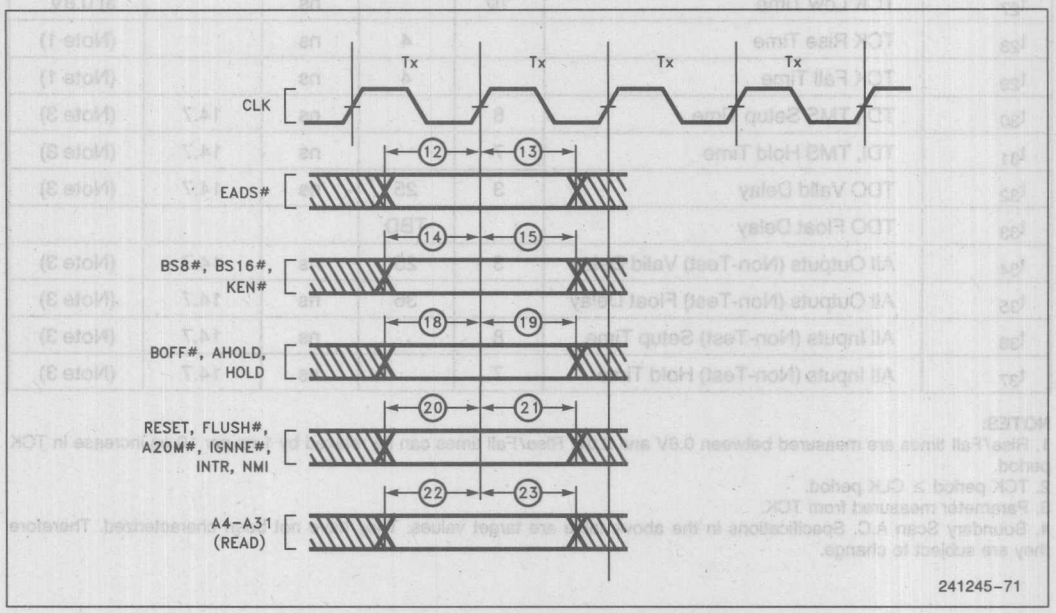


Figure 14.2. Input Setup and Hold Timing

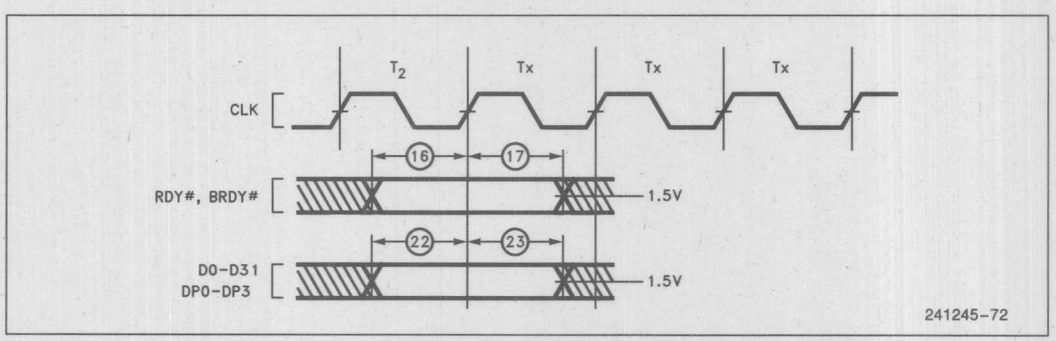


Figure 14.3. Input Setup and Hold Timing

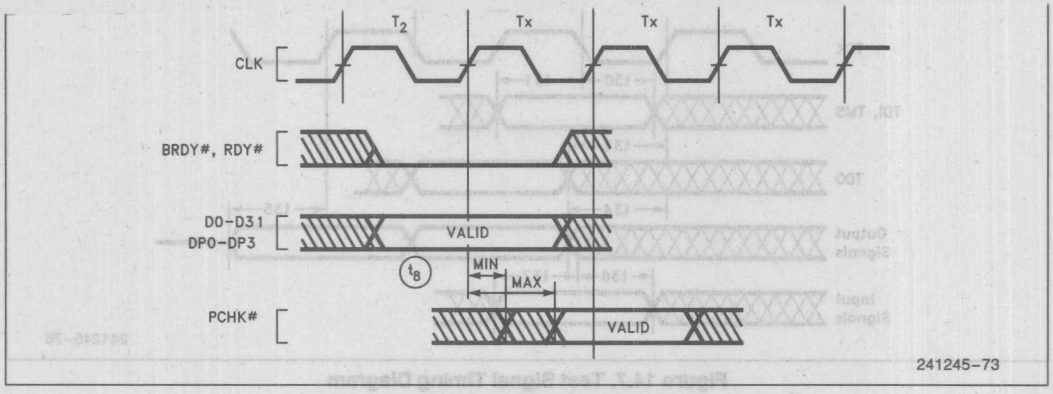


Figure 14.4. PCHK# Valid Delay Timing

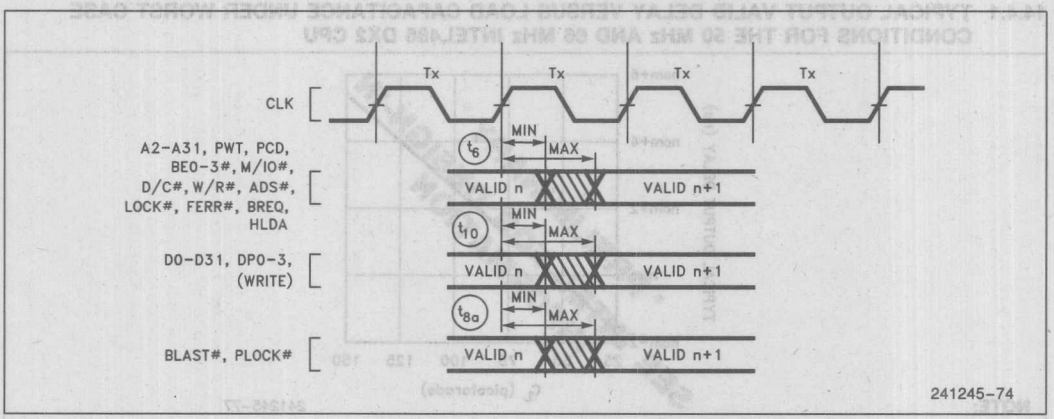


Figure 14.5. Output Valid Delay Timing

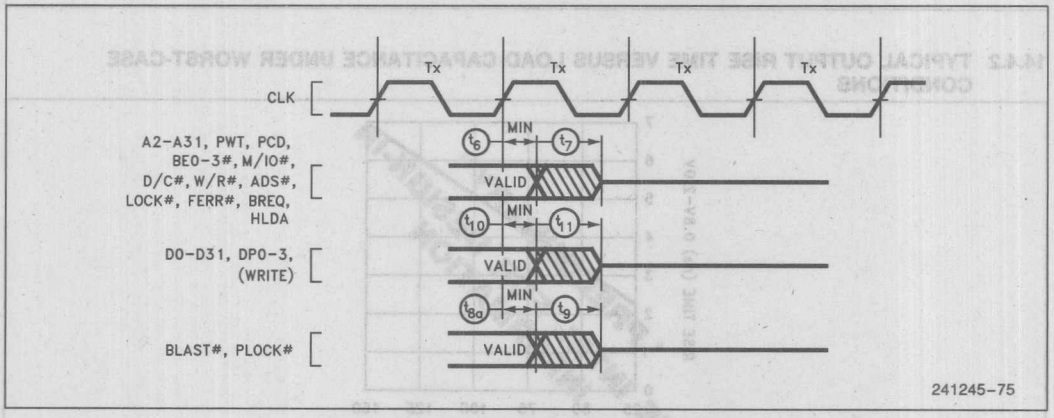


Figure 14.6. Maximum Float Delay Timing

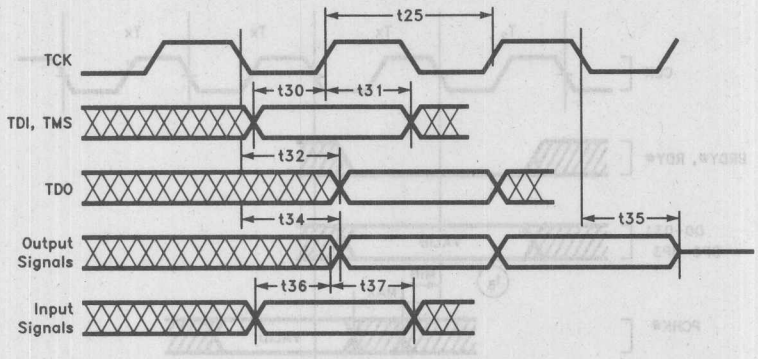
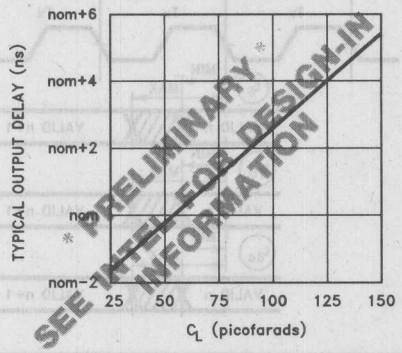


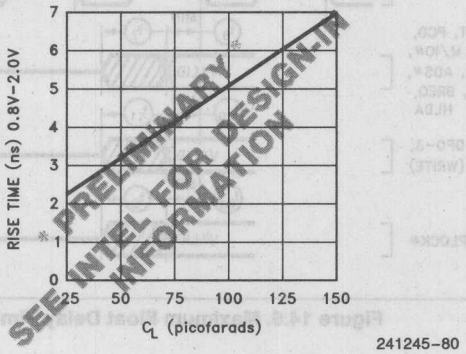
Figure 14.7. Test Signal Timing Diagram

14.4.1 TYPICAL OUTPUT VALID DELAY VERSUS LOAD CAPACITANCE UNDER WORST CASE CONDITIONS FOR THE 50 MHz AND 66 MHz INTEL486 DX2 CPU



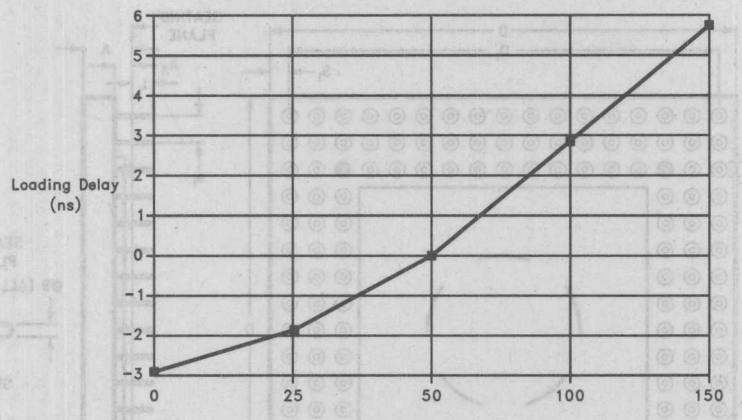
NOTE:
This graph will not be linear outside of the C_L range shown.
nom = nominal value given in A.C. Characteristics table.

14.4.2 TYPICAL OUTPUT RISE TIME VERSUS LOAD CAPACITANCE UNDER WORST-CASE CONDITIONS



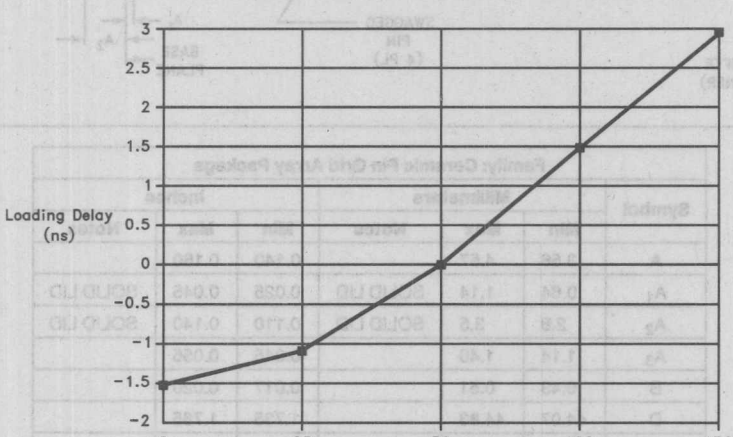
NOTE:
This graph will not be linear outside of the C_L range shown.

14.4.3.a TYPICAL LOADING DELAY VERSUS CAPACITIVE LOADING UNDER WORST-CASE CONDITIONS FOR A HIGH TO LOW TRANSITION ON THE INTEL486 DX2 CPU



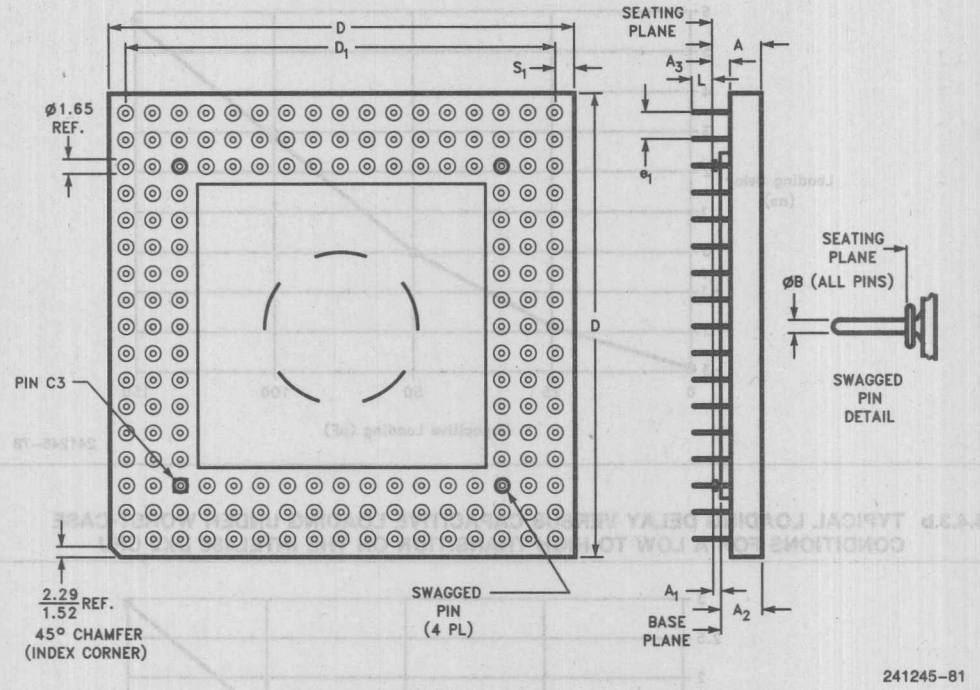
241245-78

14.4.3.b TYPICAL LOADING DELAY VERSUS CAPACITIVE LOADING UNDER WORST-CASE CONDITIONS FOR A LOW TO HIGH TRANSITION ON THE INTEL486 DX2 CPU



241245-79

15.0 MECHANICAL DATA



Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		0.140	0.180	
A_1	0.64	1.14	SOLID LID	0.025	0.045	SOLID LID
A_2	2.8	3.5	SOLID LID	0.110	0.140	SOLID LID
A_3	1.14	1.40		0.045	0.055	
B	0.43	0.51		0.017	0.020	
D	44.07	44.83		1.735	1.765	
D_1	40.51	40.77		1.595	1.605	
e_1	2.29	2.79		0.090	0.110	
L	2.54	3.30		0.100	0.130	
N	168			168		
S_1	1.52	2.54		0.060	0.100	

Figure 15.1. 168 Lead Ceramic PGA Package Dimensions

Table 15.1. Ceramic PGA Package Dimension Symbols

Letter or Symbol	Description of Dimensions
A	Distance from seating plane to highest point of body
A_1	Distance between seating plane and base plane (lid)
A_2	Distance from base plane to highest point of body
A_3	Distance from seating plane to bottom of body
B	Diameter of terminal lead pin
D	Largest overall package dimension of length
D_1	A body length dimension, outer lead center to outer lead center
e_1	Linear spacing between true lead position centerlines
L	Distance from seating plane to end of lead
S_1	Other body dimension, outer lead center to edge of body

NOTES:
1. Controlling dimension: millimeter.
2. Dimension "e_1" ("e") is non-cumulative.
3. Seating plane (standoff) is defined by P.C. board hole size: 0.0415–0.0430 inch.
4. Dimensions "B", "B_1" and "C" are nominal.
5. Details of Pin 1 identifier are optional.

15.1 Package Thermal Specifications

The Intel486 DX2 microprocessor is specified for operation when T_C (the case temperature) is within the range of 0°C–85°C. T_C may be measured in any environment to determine whether the Intel486 DX2 microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

The ambient temperature (T_A) is guaranteed as long as T_C is not violated. The ambient temperature can be calculated from θ_{JC} and θ_{JA} from the following equations.

$$T_J = T_C + P * \theta_{JC}$$
$$T_A = T_J - P * \theta_{JA}$$
$$T_A = T_C - (P * \theta_{CA})$$
$$T_C = T_A + P * [\theta_{JA} - \theta_{JC}]$$

where T_J, T_A, T_C = Junction, Ambient and Case Temperature respectively. θ_{JC}, θ_{JA} = Junction-to-Case and Junction-to-Ambient Thermal Resistance, respectively.

P = Maximum Power Consumption

The values for θ_{JA} and θ_{JC} are given in Table 13.2 for the 1.75 sq. in., 168-pin, ceramic PGA.

Table 13.3 shows the T_A allowable (without exceeding T_C) at various airflows and operating frequencies (f_{CLK}).

Note that T_A is greatly improved by attaching "fins" or a "heat sink" to the package. P (the maximum power consumption) is calculated by using the maximum I_{CC} at 5V as tabulated in the *DC Characteristics* of Section 14.

Table 15.2. Thermal Resistance (°C/W) θ_{JC} and θ_{CA} for the 50 MHz and 66 MHz Intel486™ DX2 CPU

	θ_{JC}	θ_{CA} vs Airflow—ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
With Heat Sink*	2.5	10.5	7.0	4.5	3.5	3.0	2.5
Without Heat Sink	2.0	16	14.0	10.5	9.0	8.0	7.5

*0.350" high omnidirectional heat sink (Al alloy 6063, 40 mil fin width, 155 mil center-to-center fin spacing).

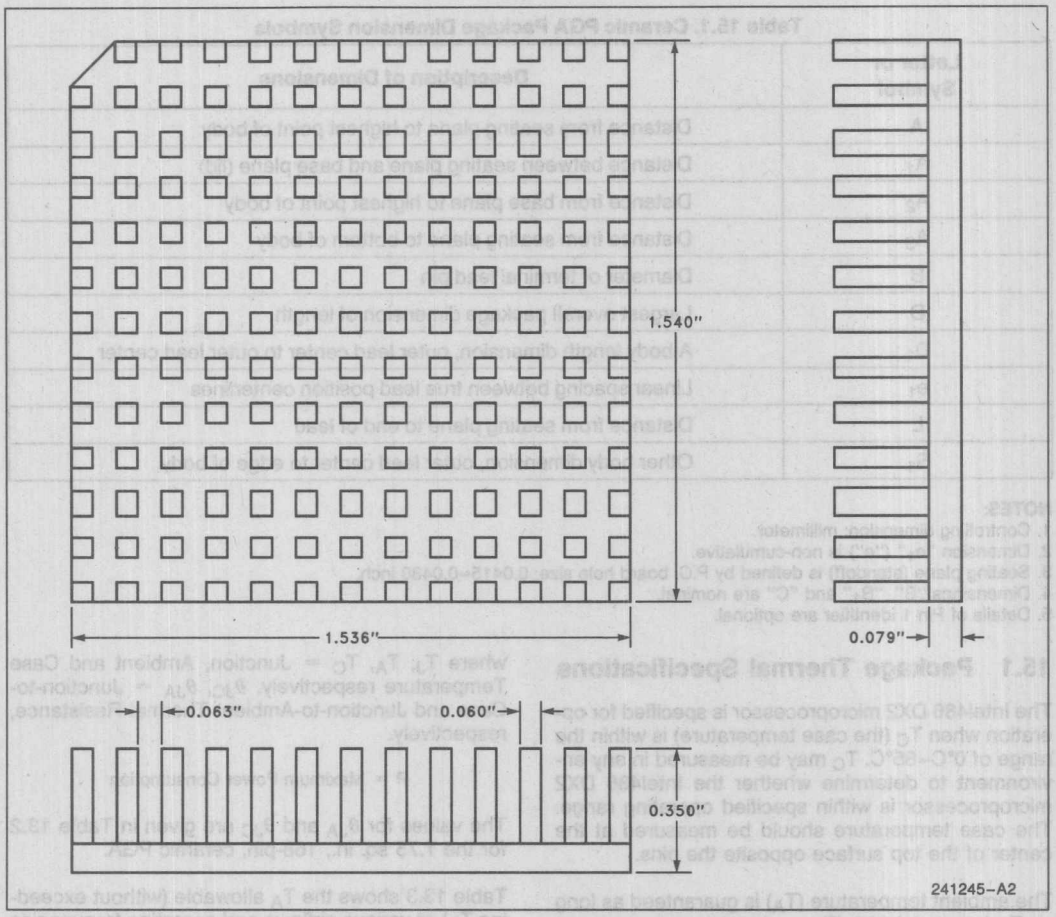


Table 15.3. Maximum T_A at Various Airflows In °C

		Airflow-ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
T_A with Heat Sink	50 MHz	32.6	50.0	62.5	67.5	70.0	72.5
	66 MHz	18.9	40.9	56.7	62.9	66.1	69.3
T_A without Heat Sink	50 MHz	5.2	15.1	32.6	40.1	45.1	47.6
	66 MHz	−15.8	−3.2	18.9	28.3	34.6	37.8

16.0 SUGGESTED SOURCES FOR INTEL486™ DX2 ACCESSORIES

Following are some suggested sources of accessories for the Intel486 DX2. They are not an endorsement of any kind, nor a warranty of the performance of any of the listed products and/or companies.

Sockets

1. McKenzie Technology
   44370 Old Palmspring Blvd.
   Fremont, CA 94538
   Tel: (415) 651-2700
2. E-CAM Technology, Inc.
   14455 North Hayden Rd.
   Suite 208
   Scottsdale, AZ 85260
   Tel: (602) 443-1949
3. Augat Inc. (for sockets with decaps)
   Interconnection Products Group
   33 Perry Ave.
   P.O. Box 779
   Attleboro, MA 02703
   Tel: (508) 222-2202

Heat Sinks/Fins

1. AAVID Engineering, Inc.
   One Kool Path
   P.O. Box 400
   Laconia, NH 03247
   Tel: (603) 528-3400

TTL Crystals/Oscillators

1. NFL Frequency Controls, Inc.
   357 Beloit Street
   Burlington, WI 53105
   Tel: (414) 763-3591
2. M-Tron
   P.O. Box 630
   Yankton, SD 57078
   Tel: (605) 665-9321

Debugging Tower

1. Emulation Technology
   2344 Walsh Ave., Building F
   Santa Clara, CA 95051
   Tel: (408) 982-0664

17.0 REVISION HISTORY

Revision -003 of the Intel486 DX2 Microprocessor Data Book contains many updates and improvements to the original version. A revision summary of major changes is listed below:

PGA Pin Table Pins B10 and C12 are no-connects

Section 6.5 Added clarification for the built in self test (BIST) during reset.

Section 7.2.9 Added explanation of bus hold and hold acknowledge protocol.

Figure 7.26b Added figure to illusrate HOLD request acknowledge during BOFF#.

Section 12.0 Replaced references to the Upgrade Socket with the Pentium OverDrive Processor Socket.

Section 12.2.2 Modified section to reflect changes in physical dimensions of heat sink unit.

Section 12.2.3 Modified End User Easy section.

Section 12.2.4 Changed section to only show ZIF socket vendors.

Section 12.3 Modified section to reflect maximum T_{SINK} and I_{CC} specification changes.

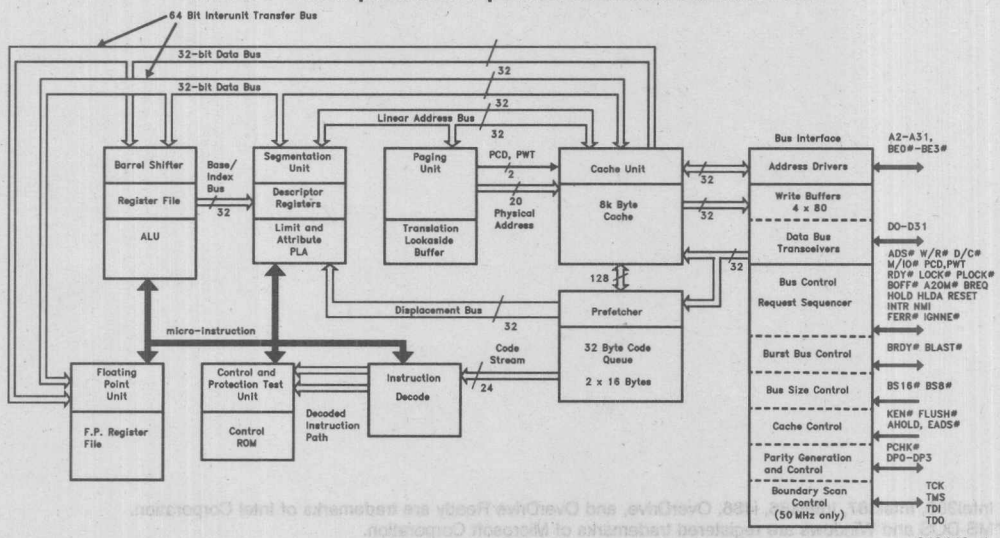
Section 12.6 Changed Pentium OverDrive Processor Socket pinout and pin table to reflect changes in orientation and NC pins.

- **Binary Compatible with Large Software Base**
  - **— MS-DOS*, OS/2**, Windows***
  - **— UNIX*** System V/386**
  - **— iRMX®, iRMK Kernels**
- **High Integration Enables On-Chip**
  - **— 8 Kbyte Code and Data Cache**
  - **— Floating Point Unit**
  - **— Paged, Virtual Memory Management**
- **Easy To Use**
  - **— Built-In Self Test**
  - **— Hardware Debugging Support**
  - **— Intel Software Support**
  - **— Extensive Third Party Software Support**
- **IEEE 1149.1 Boundary Scan Compatibility**
  - **— Available on 50 MHz Version Only**
- **Upgradable to Intel OverDrive™ Processor**
- **168-Pin Grid Array Package**
- **High Performance Design**
  - **— RISC Integer Core with Frequent Instructions Executing in One Clock**
  - **— 25 MHz, 33 MHz, and 50 MHz Clock**
  - **— 80, 106, 160 Mbyte/sec Burst Bus**
  - **— CHMOS IV and CHMOS V Process Technology**
  - **— Dynamic Bus Sizing for 8-, 16-, and 32-Bit Busses**
- **Complete 32-Bit Architecture**
  - **— Address and Data Busses**
  - **— Registers**
  - **— 8-, 16- and 32-Bit Data Types**
- **Multiprocessor Support**
  - **— Multiprocessor Instructions**
  - **— Cache Consistency Protocols**
  - **— Support for Second Level Cache**

2

The Intel486 CPU offers the highest performance for DOS, OS/2, Windows, and UNIX System V/386 applications. It is 100% binary compatible with the Intel386™ CPU. Over one million transistors integrate the RISC integer core, 8 Kbyte cache memory, floating point hardware, and memory management on-chip while retaining binary compatibility with previous members of the Intel386/Intel486 architectural family. The RISC integer core executes frequently-used instructions in one cycle, providing leadership performance levels. An 8 Kbyte unified code and data cache allow the high performance levels to be sustained. A 160 MByte/sec burst bus at 50 MHz ensures high system throughput even with inexpensive DRAMs.

Intel486™ Microprocessor Pipelined 32-Bit Microarchitecture

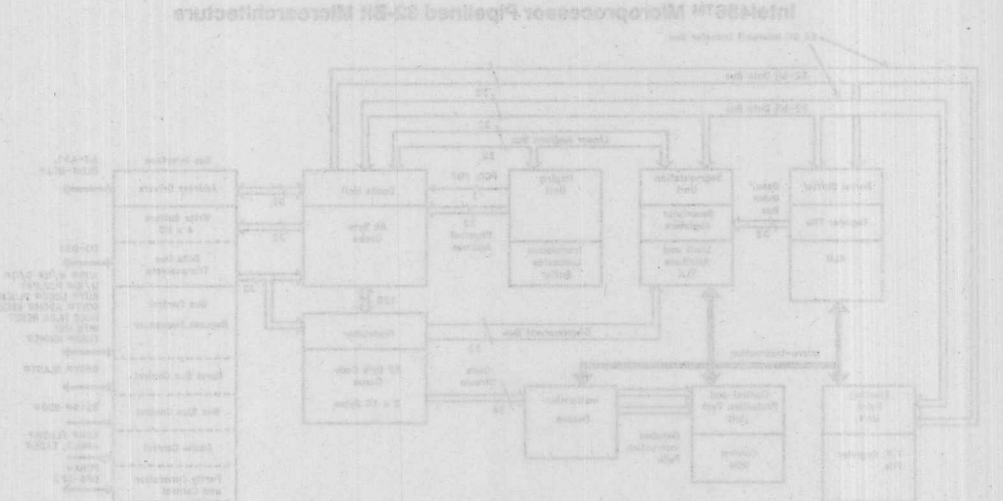


240440-1

New features enhance multiprocessing systems; new instructions speed manipulation of memory-based semaphores; and on-chip hardware ensures cache consistency and provides hooks for multilevel caches.

The built-in self-test extensively tests on-chip logic, cache memory, and the on-chip paging translation cache. Debug features include breakpoint traps on code execution and data accesses.

The Intel OverDrive Processor provides optional overall performance upgrade capability for users who want to increase their system performance up to 70% on DOS, Windows, OS/2 and Unix applications.



Intel386, Intel387, Intel486, i486, OverDrive, and OverDrive Ready are trademarks of Intel Corporation.
*MS-DOS and Windows are registered trademarks of Microsoft Corporation.
**OS/2 is a trademark of International Business Machines Corporation.
***UNIX is a trademark of UNIX Systems Laboratories.

Intel486™ MICROPROCESSOR

CONTENTS

PAGE

1.0 TABLE OF CONTENTS 2-213
Pinout 2-217
Quick Pin Reference 2-225
Component and Revision ID 2-230

2.0 ARCHITECTURAL OVERVIEW 2-231
2.1 Register Set 2-231
2.1.1 Base Architecture Registers 2-232
2.1.2 System Level Registers 2-236
2.1.3 Floating Point Registers 2-240
2.1.4 Debug and Test Registers .. 2-247
2.1.5 Register Accessibility 2-247
2.1.6 Compatibility 2-248
2.2 Instruction Set 2-249
2.3 Memory Organization 2-249
2.3.1 Address Spaces 2-249
2.3.2 Segment Register Usage ... 2-250
2.4 I/O Space 2-250
2.5 Addressing Modes 2-251
2.5.1 Addressing Modes Overview 2-251
2.5.2 Register and Immediate Modes 2-251
2.5.3 32-Bit Memory Addressing Modes 2-251
2.5.4 Differences between 16- and 32-Bit Addresses 2-253
2.6 Data Formats 2-253
2.6.1 Data Types 2-253
2.6.2 Little Endian vs Big Endian Data Formats 2-257
2.7 Interrupts 2-257
2.7.1 Interrupts and Exceptions ... 2-257
2.7.2 Interrupt Processing 2-257
2.7.3 Maskable Interrupt 2-258
2.7.4 Non-Maskable Interrupt 2-259
2.7.5 Software Interrupts 2-259
2.7.6 Interrupt and Exception Priorities 2-259
2.7.7 Instruction Restart 2-260

CONTENTS

PAGE

2.7.8 Double Fault 2-260
2.7.9 Floating Point Interrupt Vectors 2-260

3.0 REAL MODE ARCHITECTURE 2-262
3.1 Real Mode Introduction 2-262
3.2 Memory Addressing 2-262
3.3 Reserved Locations 2-263
3.4 Interrupts 2-263
3.5 Shutdown and Halt 2-263

4.0 PROTECTED MODE ARCHITECTURE 2-264
4.1 Introduction 2-264
4.2 Addressing Mechanism 2-264
4.3 Segmentation 2-265
4.3.1 Segmentation Introduction .. 2-265
4.3.2 Terminology 2-265
4.3.3 Descriptor Tables 2-265
4.3.4 Descriptors 2-267
4.4 Protection 2-275
4.4.1 Protection Concepts 2-275
4.4.2 Rules of Privilege 2-276
4.4.3 Privilege Levels 2-276
4.4.4 Privilege Level Transfers 2-277
4.4.5 Call Gates 2-280
4.4.6 Task Switching 2-280
4.4.7 Initialization and Transition to Protected Mode 2-281
4.4.8 Tools for Building Protected Systems 2-282
4.5 Paging 2-282
4.5.1 Paging Concepts 2-282
4.5.2 Paging Organization 2-283
4.5.3 Page Level Protection (R/W, U/S Bits) 2-284
4.5.4 Page Cacheability (PWT, PCD Bits) 2-285
4.5.5 Translation Lookaside Buffer 2-285
4.5.6 Paging Operation 2-286
4.5.7 Operating System Responsibilities 2-287

CONTENTS

PAGE

4.6 Virtual 8389 Environment 2-287
4.6.1 Executing 8389 Programs ... 2-287
4.6.2 Virtual 8389 Addressing Mechanism 2-287
4.6.3 Paging in Virtual Mode 2-287
4.6.4 Protection and Virtual 8389 Mode to I/O Permission Bitmap 2-288
4.6.5 Interrupt Handling 2-289
4.6.6 Entering and Leaving Virtual 8389 Mode 2-290

5.0 ON-CHIP CACHE 2-293
5.1 Cache Organization 2-293
5.2 Cache Control 2-294
5.3 Cache Line Fills 2-294
5.4 Cache Line Invalidations 2-295
5.5 Cache Replacement 2-295
5.6 Page Cacheability 2-296
5.7 Cache Flushing 2-297
5.8 Caching Translation Lookaside Buffer Entries 2-297

6.0 HARDWARE INTERFACE 2-298
6.1 Introduction 2-298
6.2 Signal Descriptions 2-299
6.2.1 Clock (CLK) 2-299
6.2.2 Address Bus (A31–A2, BE0#–BE3#) 2-299
6.2.3 Data Lines (D31–D0) 2-300
6.2.4 Parity 2-300
Data Parity Input/Outputs (DP0–DP3) 2-300
Parity Status Output (PCHK#) 2-300
6.2.5 Bus Cycle Definition 2-300
M/IO#, D/C#, W/R# Outputs 2-300
Bus Lock Output (LOCK#) 2-300
Pseudo-Lock Output (PLOCK#) 2-301

CONTENTS

PAGE

6.2.6 Bus Control 2-301
Address Status Output (ADS#) 2-301
Non-Burst Ready Input (RDY#) 2-301
6.2.7 Burst Control 2-301
Burst Ready Input (BRDY#) 2-301
Burst Last Output (BLAST#) 2-302
6.2.8 Interrupt Signals 2-302
Reset Input (RESET) 2-302
Maskable Interrupt Request Input (INTR) 2-302
Non-Maskable Interrupt Request Input (NMI) 2-302
6.2.9 Bus Arbitration Signals 2-302
Bus Request Output (BREQ) 2-302
Bus Hold Request Input (HOLD) 2-302
Bus Hold Acknowledge Output (HLDA) 2-303
Backoff Input (BOFF#) 2-303
6.2.10 Cache Invalidation 2-303
Address Hold Request Input (AHOLD) 2-303
External Address Valid Input (EADS#) 2-303
6.2.11 Cache Control 2-304
Cache Enable Input (KEN#) 2-304
Cache Flush Input (FLUSH#) 2-304
6.2.12 Page Cacheability Outputs (PWT, PCD) 2-304
6.2.13 Numeric Error Reporting ... 2-304
Floating Point Error Output (FERR#) 2-304
Ignore Numeric Error Input (IGNNE#) 2-305
6.2.14 Bus Size Control (BS16#, BS8#) 2-305

CONTENTS

PAGE

6.2.15 Address Bit 20 Mask (A20M#) 2-305
6.2.16 Boundary Scan Test Signals 2-305
Test Clock (TCK) 2-305
Test Mode Select (TMS) 2-305
Test Data Input (TDI) 2-306
Test Data Output (TDO) 2-306
6.3 Write Buffers 2-306
6.3.1 Write Buffers and I/O Cycles 2-307
6.3.2 Write Buffers Implications on Locked Bus Cycles 2-307
6.4 Interrupt and Non-Maskable Interrupt Interface 2-307
6.4.1 Interrupt Logic 2-307
6.4.2 NMI Logic 2-308
6.5 Reset and Initialization 2-308
6.5.1 Pin State during Reset 2-309

7.0 BUS OPERATION 2-311
7.1 Data Transfer Mechanism 2-311
7.1.1 Memory and I/O Spaces 2-311
7.1.2 Memory and I/O Space Organization 2-312
7.1.3 Dynamic Data Bus Sizing 2-313
7.1.4 Interfacing with 8-, 16- and 32-bit Memories 2-314
7.1.5 Dynamic Bus Sizing during Cache Line Fills 2-316
7.1.6 Operand Alignment 2-316
7.2 Bus Functional Description 2-317
7.2.1 Non-Cacheable Non-Burst Single Cycle 2-317
7.2.2 Multiple and Burst Cycle Bus Transfers 2-318
7.2.3 Cacheable Cycles 2-322
7.2.4 Burst Mode Details 2-325
7.2.5 8- and 16-Bit Cycles 2-329
7.2.6 Locked Cycles 2-331
7.2.7 Pseudo-Locked Cycles 2-332
7.2.8 Invalidate Cycles 2-332
7.2.9 Bus Hold 2-336
7.2.10 Interrupt Acknowledge 2-336

CONTENTS

PAGE

7.2.11 Special Bus Cycles 2-339
7.2.12 Bus Cycle Restart 2-340
7.2.13 Bus States 2-341
7.2.14 Floating Point Error Handling 2-342
7.2.15 Floating Point Error Handling in AT Compatible Systems 2-342

8.0 TESTABILITY 2-344
8.1 Built-In Self Test (BIST) 2-344
8.2 On-Chip Cache Testing 2-344
8.2.1 Cache Testing Registers TR3, TR4 and TR5 2-345
Cache Data Test Register: TR3 2-345
Cache Status Test Register: TR4 2-345
Cache Control Test Register: TR5 2-345
8.2.2 Cache Testability Write 2-345
8.2.3 Cache Testability Read 2-347
8.2.4 Flush Cache 2-347
8.3 Translation Lookaside Buffer (TLB) Testing 2-347
8.3.1 Translation Lookaside Buffer Organization 2-347
8.3.2 TLB Test Registers: TR6 and TR7 2-348
Command Test Register: TR6 2-349
Data Test Register: TR7 2-349
8.3.3 TLB Write Test 2-350
8.3.4 TLB Lookup Test 2-350
8.4 Tristate Output Test Mode 2-350
8.5 Intel486™ Microprocessor Boundary Scan (JTAG) 2-350
8.5.1 Boundary Scan Architecture 2-351
8.5.2 Data Registers 2-351
8.5.3 Instruction Register 2-352
8.5.4 Test Access Port (TAP) Controller 2-354
8.5.5 Boundary Scan Register Cell 2-356
8.5.6 TAP Controller Initialization 2-357
8.5.7 Boundary Scan Description Language (BSDL) 2-357

CONTENTS

PAGE

9.0 DEBUGGING SUPPORT 2-358
9.1 Breakpoint Instructions 2-358
9.2 Single Step Instructions 2-358
9.3 Debug Registers 2-358
9.3.1 Linear Address Breakpoint Registers 2-358
9.3.2 Debug Control Register 2-358
9.3.3 Debug Status Register 2-361
9.3.4 Use of Resume Flag (RF) in Flag Register 2-361

10.0 INSTRUCTION SET SUMMARY .. 2-362
10.1 Intel486™ Microprocessor Instruction Encoding and Clock Count Summary 2-362
10.2 Instruction Encoding 2-381
10.2.1 Overview 2-381
10.2.2 32-Bit Extensions of the Instruction Set 2-382
10.2.3 Encoding of Integer Instruction Fields 2-382
10.2.4 Encoding of Floating Point Instruction Fields 2-388

11.0 DIFFERENCES WITH THE 386 MICROPROCESSOR 2-389

12.0 OVERDRIVE PROCESSOR SOCKET 2-390
12.1 OverDrive Processor Overview 2-390
12.1.1 Hardware Interface 2-390
12.1.2 Testability 2-391
12.1.3 Instruction Set Summary ... 2-391
12.2 Intel OverDrive Processor Circuit Design 2-393
12.2.1 Upgrade Circuit for PGA Intel486 DX Based Systems 2-393
12.3 Socket Layout 2-394
12.3.1 Physical Dimensions 2-394
12.3.2 "End User Easy" Upgradability 2-397
12.3.3 ZIF and LIF Socket Vendors 2-398

CONTENTS

PAGE

12.4 Thermal Management 2-398
12.4.1 Thermal Calculations for Hypothetical System 2-398
12.4.2 OverDrive Heat Sinks 2-399
12.5 BIOS and Software 2-399
12.5.1 Intel OverDrive Processor Detection 2-399
12.5.2 Timing Dependent Loops .. 2-400
12.6 OverDrive Processor Socket Pinout 2-401
12.7 D.C./A.C. Specifications 2-404

13.0 ELECTRICAL DATA 2-405
13.1 Power and Grounding 2-405
13.2 Maximum Ratings 2-405
13.3 D.C. Specifications 2-406
13.4 A.C. Specifications 2-407
13.5 Designing for ICD-486 2-416

14.0 MECHANICAL DATA 2-420
14.1 Package Thermal Specifications 2-421

15.0 LOW POWER Intel486™ DX MICROPROCESSOR 2-423
15.1 Introduction 2-423
15.2 Pinout 2-425
15.3 Pin Cross Reference (Intel486™ DX CPU) 2-427
15.4 Pin Description 2-427
15.5 Signal Description 2-428
15.6 Architecture Overview 2-431
15.7 Variable CPU Frequency 2-431
15.8 D.C./A.C. Specifications 2-433
15.8.1 D.C. Specifications 2-433
15.8.2 Power Supply Current vs Frequency 2-434
15.8.3 A.C. Specifications 2-434

16.0 SUGGESTED SOURCES FOR Intel486™ ACCESSORIES 2-437

17.0 REVISION HISTORY 2-438

APPENDIX A 2-441

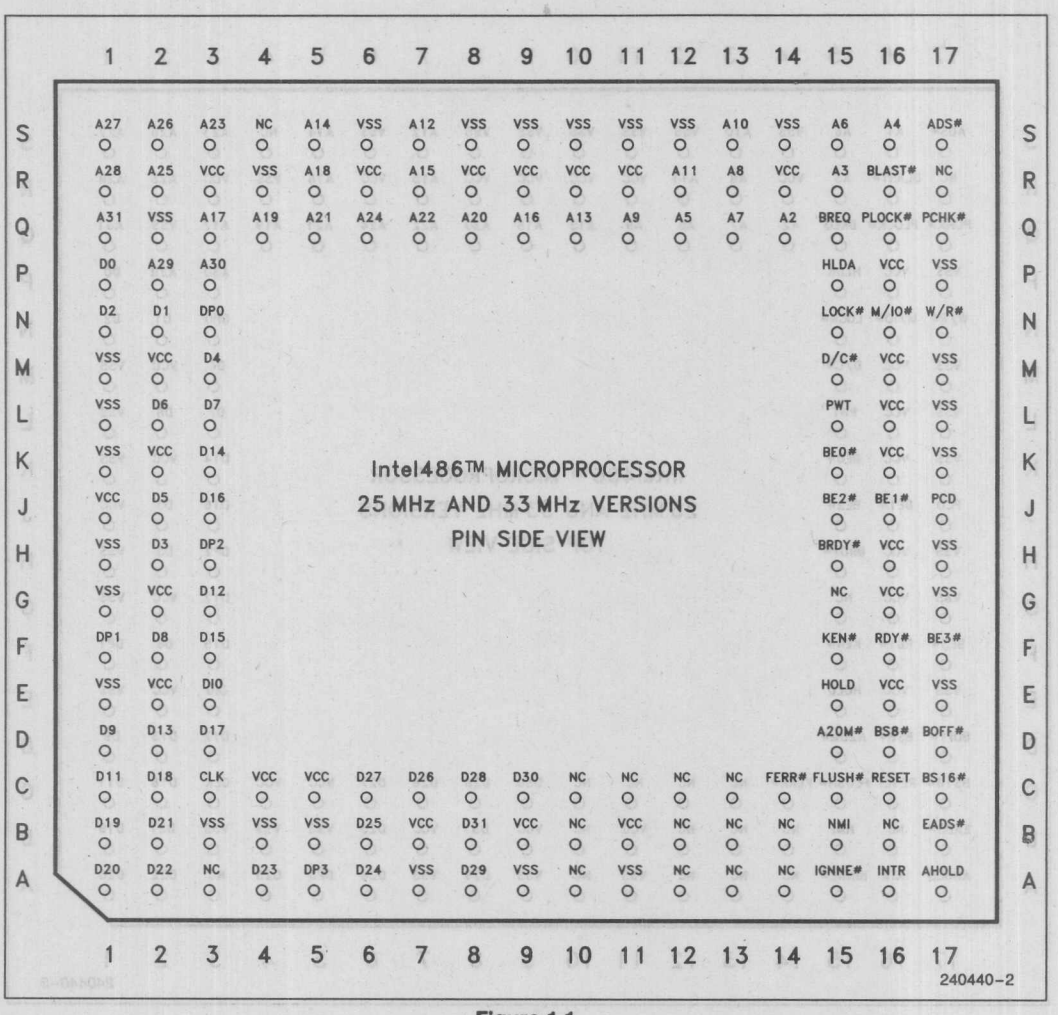

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
S A27 A26 A23 NC A14 VSS A12 VSS VSS VSS VSS VSS A10 VSS A6 A4 ADS#
R A28 A25 VCC VSS A18 VCC A15 VCC VCC VCC VCC A11 A8 VCC A3 BLAST# NC
Q A31 VSS A17 A19 A21 A24 A22 A20 A16 A13 A9 A5 A7 A2 BREQ PLOCK# PCHK#
P D0 A29 A30 HLDA VCC VSS
N D2 D1 DP0 LOCK# M/IO# W/R#
M VSS VCC D4 D/C# VCC VSS
L VSS D6 D7 PWT VCC VSS
K VSS VCC D14 BE0# VCC VSS
J VCC D5 D16 BE2# BE1# PCD
H VSS D3 DP2 BRDY# VCC VSS
G VSS VCC D12 NC VCC VSS
F DP1 D8 D15 KEN# RDY# BE3#
E VSS VCC DI0 HOLD VCC VSS
D D9 D13 D17 A20M# BS8# BOFF#
C D11 D18 CLK VCC VCC D27 D26 D28 D30 NC NC NC NC FERR# FLUSH# RESET BS16#
B D19 D21 VSS VSS VSS D25 VCC D31 VCC NC VCC NC NC NC NMI NC EADS#
A D20 D22 NC D23 DP3 D24 VSS D29 VSS NC VSS NC NC NC IGNNE# INTR AHOLD
Intel486™ MICROPROCESSOR
25 MHz AND 33 MHz VERSIONS
PIN SIDE VIEW
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
240440-2

Figure 1.1

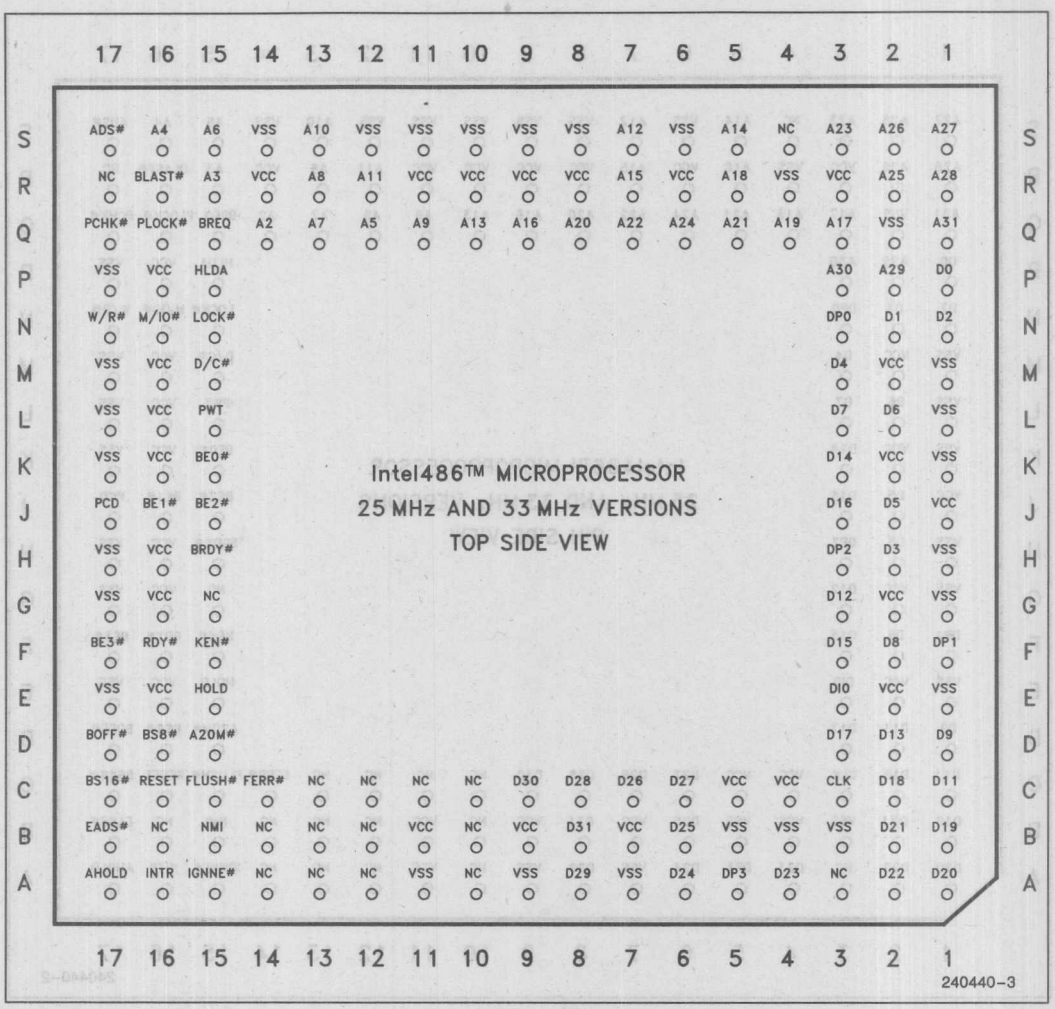

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
S ADS# A4 A6 VSS A10 VSS VSS VSS VSS VSS A12 VSS A14 NC A23 A26 A27
R NC BLAST# A3 VCC A8 A11 VCC VCC VCC VCC A15 VCC A18 VSS VCC A25 A28
Q PCHK# PLOCK# BREQ A2 A7 A5 A9 A13 A16 A20 A22 A24 A21 A19 A17 VSS A31
P VSS VCC HLDA A30 A29 D0
N W/R# M/IO# LOCK# DP0 D1 D2
M VSS VCC D/C# D4 VCC VSS
L VSS VCC PWT D7 D6 VSS
K VSS VCC BE0# D14 VCC VSS
J PCD BE1# BE2# D16 D5 VCC
H VSS VCC BRDY# DP2 D3 VSS
G VSS VCC NC D12 VCC VSS
F BE3# RDY# KEN# D15 D8 DP1
E VSS VCC HOLD DI0 VCC VSS
D BOFF# BS8# A20M# D17 D13 D9
C BS16# RESET FLUSH# FERR# NC NC NC NC D30 D28 D26 D27 VCC VCC CLK D18 D11
B EADS# NC NMI NC NC NC VCC NC VCC D31 VCC D25 VSS VSS VSS D21 D19
A AHOLD INTR IGNNE# NC NC NC VSS NC VSS D29 VSS D24 DP3 D23 NC D22 D20
Intel486™ MICROPROCESSOR
25 MHz AND 33 MHz VERSIONS
TOP SIDE VIEW
17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
240440-3

Figure 1.2

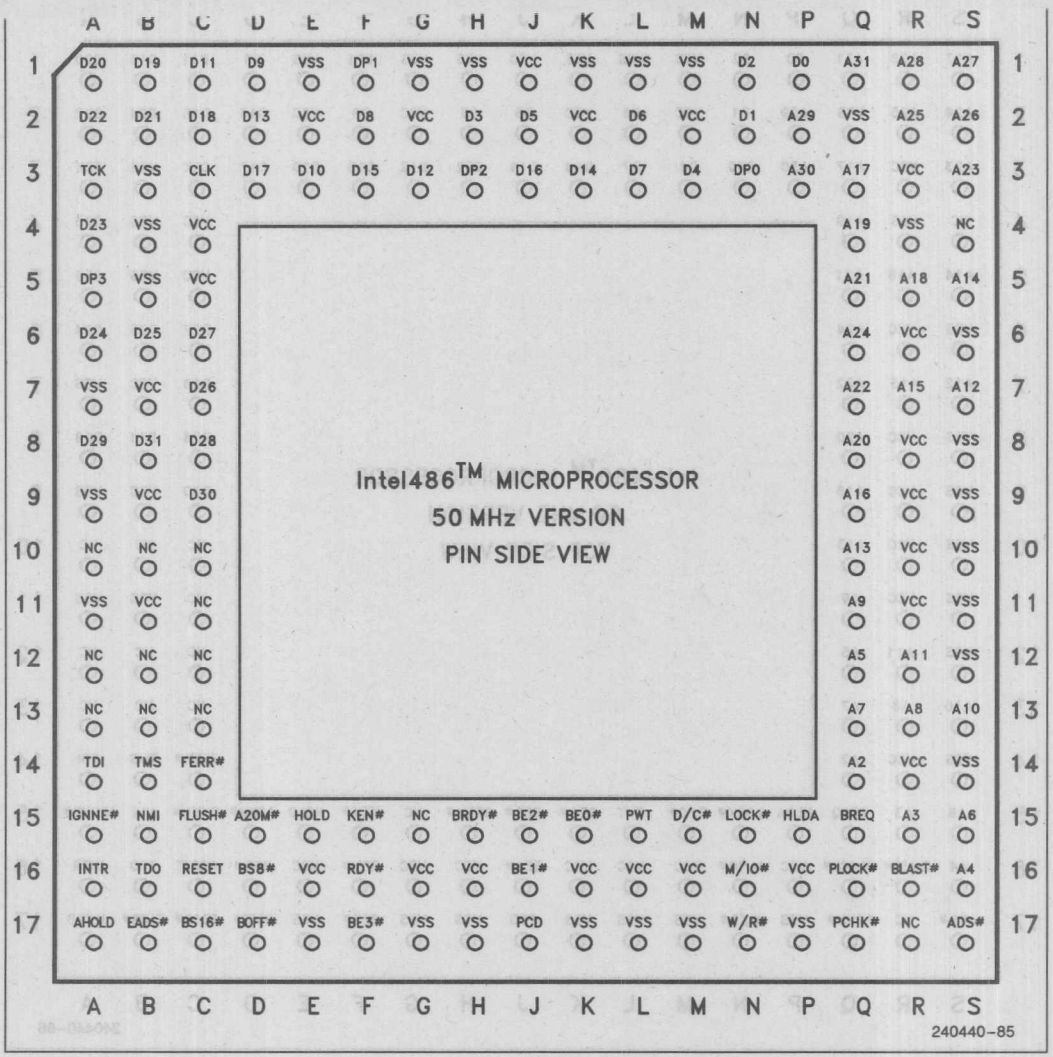

A B C D E F G H J K L M N P Q R S
1 D20 D19 D11 D9 VSS DP1 VSS VSS VCC VSS VSS VSS D2 D0 A31 A28 A27 1
2 D22 D21 D18 D13 VCC D8 VCC D3 D5 VCC D6 VCC D1 A29 VSS A25 A26 2
3 TCK VSS CLK D17 D10 D15 D12 DP2 D16 D14 D7 D4 DP0 A30 A17 VCC A23 3
4 D23 VSS VCC A19 VSS NC 4
5 DP3 VSS VCC A21 A18 A14 5
6 D24 D25 D27 A24 VCC VSS 6
7 VSS VCC D26 A22 A15 A12 7
8 D29 D31 D28 A20 VCC VSS 8
9 VSS VCC D30 A16 VCC VSS 9
10 NC NC NC A13 VCC VSS 10
11 VSS VCC NC A9 VCC VSS 11
12 NC NC NC A5 A11 VSS 12
13 NC NC NC A7 A8 A10 13
14 TDI TMS FERR# A2 VCC VSS 14
15 IGNNE# NMI FLUSH# A20M# HOLD KEN# NC BRDY# BE2# BE0# PWT D/C# LOCK# HLDA BREQ A3 A6 15
16 INTR TDO RESET BS8# VCC RDY# VCC VCC BE1# VCC VCC VCC M/IO# VCC PLOCK# BLAST# A4 16
17 AHOLD EADS# BS16# BOFF# VSS BE3# VSS VSS PCD VSS VSS VSS W/R# VSS PCHK# NC ADS# 17
A B C D E F G H J K L M N P Q R S
Intel486™ MICROPROCESSOR
50 MHz VERSION
PIN SIDE VIEW
240440-85

Figure 1.3

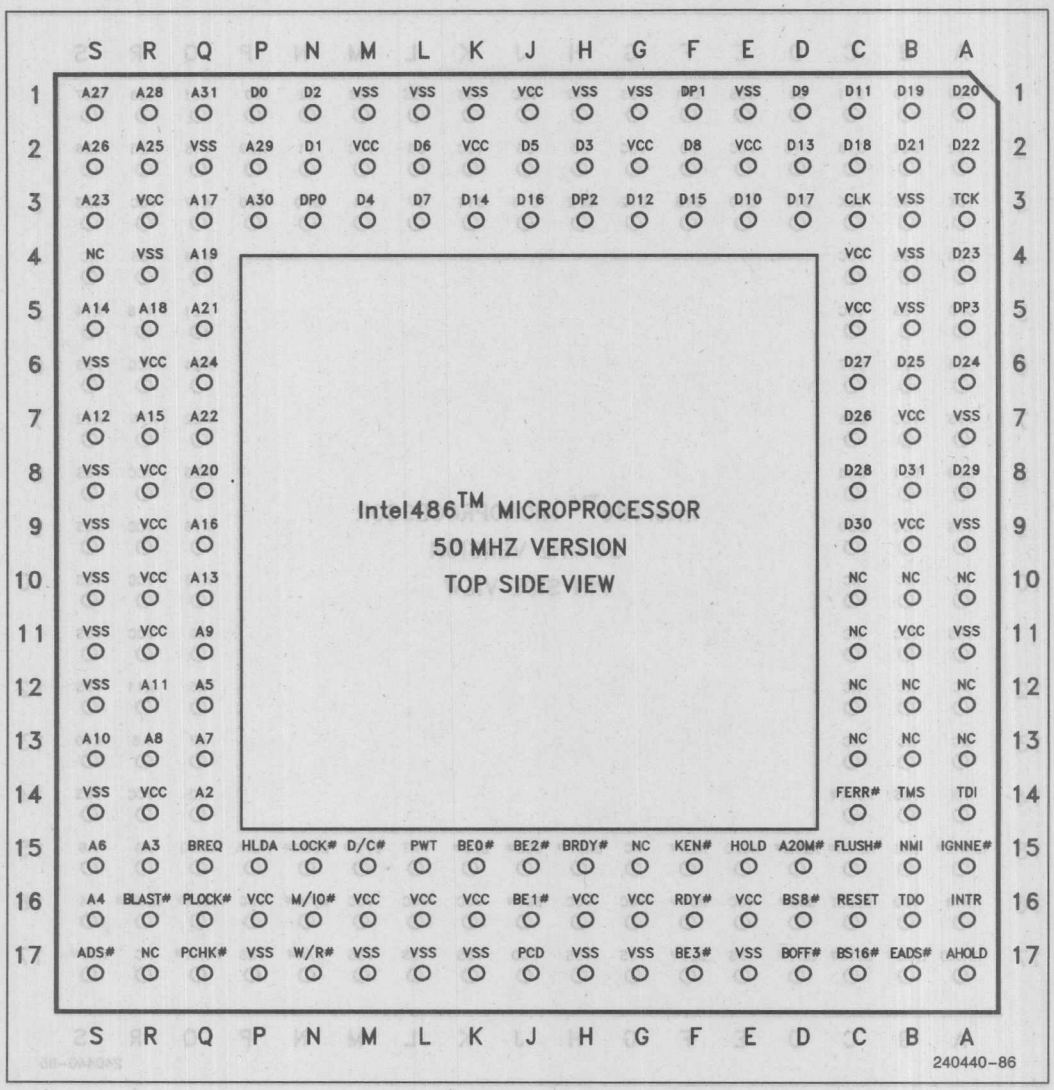

S R Q P N M L K J H G F E D C B A
1 A27 A28 A31 D0 D2 VSS VSS VSS VCC VSS VSS DP1 VSS D9 D11 D19 D20 1
2 A26 A25 VSS A29 D1 VCC D6 VCC D5 D3 VCC D8 VCC D13 D18 D21 D22 2
3 A23 VCC A17 A30 DP0 D4 D7 D14 D16 DP2 D12 D15 D10 D17 CLK VSS TCK 3
4 NC VSS A19 VCC VSS D23 4
5 A14 A18 A21 VCC VSS DP3 5
6 VSS VCC A24 D27 D25 D24 6
7 A12 A15 A22 D26 VCC VSS 7
8 VSS VCC A20 D28 D31 D29 8
9 VSS VCC A16 D30 VCC VSS 9
10 VSS VCC A13 NC NC NC 10
11 VSS VCC A9 NC VCC VSS 11
12 VSS A11 A5 NC NC NC 12
13 A10 A8 A7 NC NC NC 13
14 VSS VCC A2 FERR# TMS TDI 14
15 A6 A3 BREQ HLDA LOCK# D/C# PWT BE0# BE2# BRDY# NC KEN# HOLD A20M# FLUSH# NMI IGNNE# 15
16 A4 BLAST# PLOCK# VCC M/IO# VCC VCC VCC BE1# VCC VCC RDY# VCC BS8# RESET TDO INTR 16
17 ADS# NC PCHK# VSS W/R# VSS VSS VSS PCD VSS VSS BE3# VSS BOFF# BS16# EADS# AHOLD 17
S R Q P N M L K J H G F E D C B A
Intel486™ MICROPROCESSOR
50 MHZ VERSION
TOP SIDE VIEW
240440-86

Figure 1.4

Pin Cross Reference by Pin Name

Address		Data		Control		Test (50 MHz Only)		N/C	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	TCK	A3	A3(1)	B7	A7
A_3	R15	D_1	N2	ADS#	S17	TDI	A14	A10	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	TDO	B16	A12	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	TMS	B14	A13	C4	B3
A_6	S15	D_4	M3	BE1#	J16			A14(1)	C5	B4
A_7	Q13	D_5	J2	BE2#	J15			B10	E2	B5
A_8	R13	D_6	L2	BE3#	F17			B12	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16			B13	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17			B14(1)	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15			B16(1)	H16	G17
A_{12}	S7	D_{10}	E3	BREQ	Q15			C10	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16			C11	K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17			C12	K16	K1
A_{15}	R7	D_{13}	D2	CLK	C3			C13	L16	K17
A_{16}	Q9	D_{14}	K3	D/C#	M15			G15	M2	L1
A_{17}	Q3	D_{15}	F3	DP0	N3			R17	M16	L17
A_{18}	R5	D_{16}	J3	DP1	F1			S4	P16	M1
A_{19}	Q4	D_{17}	D3	DP2	H3				R3	M17
A_{20}	Q8	D_{18}	C2	DP3	A5				R6	P17
A_{21}	Q5	D_{19}	B1	EADS#	B17				R8	Q2
A_{22}	Q7	D_{20}	A1	FERR#	C14				R9	R4
A_{23}	S3	D_{21}	B2	FLUSH#	C15				R10	S6
A_{24}	Q6	D_{22}	A2	HLDA	P15				R11	S8
A_{25}	R2	D_{23}	A4	HOLD	E15				R14	S9
A_{26}	S2	D_{24}	A6	IGNNE#	A15					S10
A_{27}	S1	D_{25}	B6	INTR	A16					S11
A_{28}	R1	D_{26}	C7	KEN#	F15					S12
A_{29}	P2	D_{27}	C6	LOCK#	N15					S14
A_{30}	P3	D_{28}	C8	M/IO#	N16					
A_{31}	Q1	D_{29}	A8	NMI	B15					
		D_{30}	C9	PCD	J17					
		D_{31}	B8	PCHK#	Q17					
				PWT	L15					
				PLOCK#	Q16					
				RDY#	F16					
				RESET	C16					
				W/R#	N17					

NOTE:
1. These pins are no longer No-Connects on the 50 MHz version.

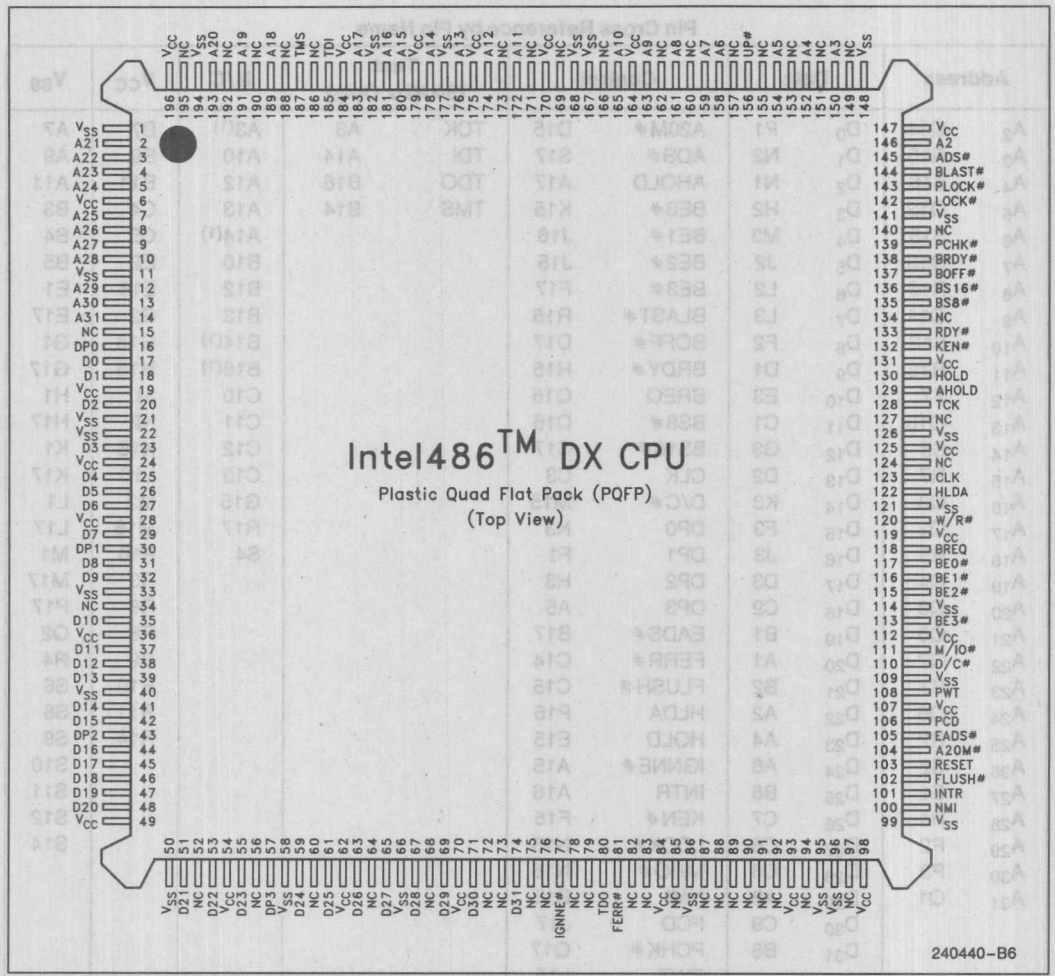


Figure 1.5. Intel486™ DX CPU 196 Lead PQFP Pinout

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	V_{SS}	50	V_{SS}	99	V_{SS}	148	V_{SS}
2	A_{21}	51	D_{21}	100	NMI	149	NC
3	A_{22}	52	NS	101	INTR	150	A_3
4	A_{23}	53	D_{22}	102	FLUSH#	151	NC
5	A_{24}	54	V_{CC}	103	RESET	152	A_4
6	V_{CC}	55	D_{23}	104	A20M#	153	NC
7	A_{25}	56	NC	105	EADS#	154	A_5
8	A_{26}	57	DP3	106	PCD	155	NC
9	A_{27}	58	V_{SS}	107	V_{CC}	156	UP#
10	A_{28}	59	D_{24}	108	PWT	157	NC
11	V_{SS}	60	NC	109	V_{SS}	158	A_6
12	A_{29}	61	D_{25}	110	D/C#	159	A_7
13	A_{30}	62	V_{CC}	111	M/IO#	160	NC
14	A_{31}	63	D_{26}	112	V_{CC}	161	A_8
15	NC	64	NC	113	BE3#	162	NC
16	DP0	65	D_{27}	114	V_{SS}	163	A_9
17	D_0	66	V_{SS}	115	BE2#	164	V_{CC}
18	D_1	67	D_{28}	116	BE1#	165	A_{10}
19	V_{CC}	68	NC	117	BE0#	166	NC
20	D_2	69	D_{29}	118	BREQ	167	V_{SS}
21	V_{SS}	70	V_{CC}	119	V_{CC}	168	V_{SS}
22	V_{SS}	71	D_{30}	120	W/R#	169	NC
23	D_3	72	NC	121	V_{SS}	170	V_{CC}
24	V_{CC}	73	NC	122	HLDA	171	NC
25	D_4	74	D_{31}	123	CLK	172	A_{11}
26	D_5	75	NC	124	NC	173	NC
27	D_6	76	NC	125	V_{CC}	174	A_{12}
28	V_{CC}	77	IGNNE#	126	V_{SS}	175	V_{CC}
29	D_7	78	NC	127	NC	176	A_{13}
30	DP1	79	NC	128	TCK	177	V_{SS}
31	D_8	80	TDO	129	AHOLD	178	A_{14}
32	D_9	81	FERR#	130	HOLD	179	V_{CC}
33	V_{SS}	82	NC	131	V_{CC}	180	A_{15}
34	NC	83	NC	132	KEN#	181	A_{16}
35	D_{10}	84	V_{CC}	133	RDY#	182	V_{SS}
36	V_{CC}	85	NC	134	NC	183	A_{17}
37	D_{11}	86	V_{SS}	135	BS8#	184	V_{CC}
38	D_{12}	87	NC	136	BS16#	185	TDI
39	D_{13}	88	NC	137	BOFF#	186	NC
40	V_{SS}	89	NC	138	BRDY#	187	TMS
41	D_{14}	90	NC	139	PCHK#	188	NC
42	D_{15}	91	NC	140	NC	189	A_{18}
43	DP2	92	NC	141	V_{SS}	190	NC
44	D_{16}	93	V_{CC}	142	LOCK#	191	A_{19}
45	D_{17}	94	NC	143	PLOCK#	192	NC
46	D_{18}	95	V_{SS}	144	BLAST#	193	A_{20}
47	D_{19}	96	V_{SS}	145	ADS#	194	V_{SS}
48	D_{20}	97	NC	146	A_2	195	NC
49	V_{CC}	98	V_{CC}	147	V_{CC}	196	V_{CC}

NC pins should always remain unconnected, for other recommendations see Section 12.1.3.

Pin Cross Reference by Signal Type (PQFP Package)

Address		Data		Control		NC	V_{CC}	V_{SS}
A_2	146	D_0	17	A20M#	104	15	6	1
A_3	150	D_1	18	ADS#	145	34	19	11
A_4	152	D_2	20	AHOLD	129	52	24	21
A_5	154	D_3	23	BE0#	117	56	28	22
A_6	158	D_4	25	BE1#	116	60	36	33
A_7	159	D_5	26	BE2#	115	64	49	40
A_8	161	D_6	27	BE3#	113	68	54	50
A_9	163	D_7	29	BLAST#	144	72	62	58
A_{10}	165	D_8	31	BOFF#	137	73	70	66
A_{11}	172	D_9	32	BRDY#	138	75	84	86
A_{12}	174	D_{10}	35	BREQ	118	76	93	95
A_{13}	176	D_{11}	37	BS#	135	78	98	96
A_{14}	178	D_{12}	38	BS16#	136	79	107	99
A_{15}	180	D_{13}	39	CLK	123	82	112	109
A_{16}	181	D_{14}	41	D/C#	110	83	119	114
A_{17}	183	D_{15}	42	DP0	16	85	125	121
A_{18}	189	D_{16}	44	DP1	30	87	131	126
A_{19}	191	D_{17}	45	DP2	43	88	147	141
A_{20}	193	D_{18}	46	DP3	57	89	164	148
A_{21}	2	D_{19}	47	EADS#	105	90	170	167
A_{22}	3	D_{20}	48	FERR#	81	91	175	168
A_{23}	4	D_{21}	51	FLUSH#	102	92	179	177
A_{24}	5	D_{22}	53	HLDA	122	94	184	182
A_{25}	7	D_{23}	55	HOLD	130	97	196	194
A_{26}	8	D_{24}	59	IGNNE#	77	124		
A_{27}	9	D_{25}	61	INTR	101	127		
A_{28}	10	D_{26}	63	KEN#	132	134		
A_{29}	12	D_{27}	65	LOCK#	142	140		
A_{30}	13	D_{28}	67	M/IO#	111	149		
A_{31}	14	D_{29}	69	NMI	100	151		
		D_{30}	71	PCD	106	153		
		D_{31}	74	PHCK#	139	155		
				PWT	108	157		
				PLOCK#	143	160		
				RDY#	133	162		
				RESET	103	166		
				TDI	185	169		
				TDO	80	171		
				TMS	187	173		
				W/R#	120	186		
						188		
						190		
						192		
						195		

NC Pins should always remain unconnected, for other recommendations see Section 12.1.3.

QUICK PIN REFERENCE

What follows is a brief pin description. For detailed signal descriptions refer to Section 6.

Symbol	Type	Name and Function
CLK	I	*Clock* provides the fundamental timing and the internal operating frequency for the Intel486 Microprocessor. All external timing parameters are specified with respect to the rising edge of CLK.
ADDRESS BUS		
A31–A4 A2–A3	I/O O	A31–A2 are the *address lines* of the microprocessor. A31–A2, together with the byte enables BE0#–BE3#, define the physical area of memory or input/output space accessed. Address lines A31–A4 are used to drive addresses into the microprocessor to perform cache line invalidations. Input signals must meet setup and hold times t_{22} and t_{23}. A31–A2 are not driven during bus or address hold.
BE0–3#	O	The *byte enable* signals indicate active bytes during read and write cycles. During the first cycle of a cache fill, the external system should assume that all byte enables are active. BE3# applies to D24–D31, BE2# applies to D16–D23, BE1# applies to D8–D16 and BE0# applies to D0–D7. BE0#–BE3# are active LOW and are not driven during bus hold.
DATA BUS		
D31–D0	I/O	These are the *data lines* for the Intel486 Microprocessor. Lines D0–D7 define the least significant byte of the data bus while lines D24–D31 define the most significant byte of the data bus. These signals must meet setup and hold times t_{22} and t_{23} for proper operation on reads. These pins are driven during the second and subsequent clocks of write cycles.
DATA PARITY		
DP0–DP3	I/O	There is one *data parity* pin for each byte of the data bus. Data parity is generated on all write data cycles with the same timing as the data driven by the Intel486 Microprocessor. Even parity information must be driven back into the microprocessor on the data parity pins with the same timing as read information to insure that the correct parity check status is indicated by the Intel486 microprocessor. The signals read on these pins do not affect program execution. Input signals must meet setup and hold times t_{22} and t_{23}. DP0–DP3 should be connected to V_{CC} through a pullup resistor in systems which do not use parity. DP0–DP3 are active HIGH and are driven during the second and subsequent clocks of write cycles.
PCHK#	O	*Parity Status* is driven on the PCHK# pin the clock after ready for read operations. The parity status is for data sampled at the end of the previous clock. A parity error is indicated by PCHK# being LOW. Parity status is only checked for enabled bytes as indicated by the byte enable and bus size signals. PCHK# is valid only in the clock immediately after read data is returned to the microprocessor. At all other times PCHK# is inactive (HIGH). PCHK# is never floated.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
BUS CYCLE DEFINITION		
M/IO# D/C# W/R#	O O O	The *memory/input-output, data/control* and *write/read* lines are the primary bus definition signals. These signals are driven valid as the ADS# signal is asserted.

M/IO#	D/C#	W/R#	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Halt/Special Cycle
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Code Read
1	0	1	Reserved
1	1	0	Memory Read
1	1	1	Memory Write

The bus definition signals are not driven during bus hold and follow the timing of the address bus. Refer to Section 7.2.11 for a description of the special bus cycles.

LOCK#	O	The *bus lock* pin indicates that the current bus cycle is locked. The Intel486 Microprocessor will not allow a bus hold when LOCK# is asserted (but address holds are allowed). LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after the last clock of the last locked bus cycle. The last locked cycle ends when ready is returned. LOCK# is active LOW and is not driven during bus hold. Locked read cycles will not be transformed into cache fill cycles if KEN# is returned active.
PLOCK#	O	The *pseudo-lock* pin indicates that the current bus transaction requires more than one bus cycle to complete. Examples of such operations are floating point long reads and writes (64 bits), segment table descriptor reads (64 bits), in addition to cache line fills (128 bits). The Intel486 Microprocessor will drive PLOCK# active until the addresses for the last bus cycle of the transaction have been driven regardless of whether RDY# or BRDY# have been returned. Normally PLOCK# and BLAST# are inverse of each other. However during the first bus cycle of a 64-bit floating point write, both PLOCK# and BLAST# will be asserted. PLOCK# is a function of the BS8#, BS16# and KEN# inputs. PLOCK# should be sampled only in the clock ready is returned. PLOCK# is active LOW and is not driven during bus hold.
BUS CONTROL		
ADS#	O	The *address status* output indicates that a valid bus cycle definition and address are available on the cycle definition lines and address bus. ADS# is driven active in the same clock as the addresses are driven. ADS# is active LOW and is not driven during bus hold.
RDY#	I	The *non-burst ready* input indicates that the current bus cycle is complete. RDY# indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted data from the Intel486 Microprocessor in response to a write. RDY# is ignored when the bus is idle and at the end of the first clock of the bus cycle. RDY# is active during address hold. Data can be returned to the processor while AHOLD is active. RDY# is active LOW, and is not provided with an internal pullup resistor. RDY# must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
BURST CONTROL		
BRDY#	I	The *burst ready input* performs the same function during a burst cycle that RDY# performs during a non-burst cycle. BRDY# indicates that the external system has presented valid data in response to a read or that the external system has accepted data in response to a write. BRDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle. BRDY# is sampled in the second and subsequent clocks of a burst cycle. The data presented on the data bus will be strobed into the microprocessor when BRDY# is sampled active. If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is prematurely aborted. BRDY# is active LOW and is provided with a small pullup resistor. BRDY# must satisfy the setup and hold times t_{16} and t_{17}.
BLAST#	O	The *burst last* signal indicates that the next time BRDY# is returned the burst bus cycle is complete. BLAST# is active for both burst and non-burst bus cycles. BLAST# is active LOW and is not driven during bus hold.
INTERRUPTS		
RESET	I	The *reset* input forces the Intel486 Microprocessor to begin execution at a known state. The microprocessor cannot begin execution of instructions until at least 1 ms after V_{CC} and CLK have reached their proper DC and AC specifications. The RESET pin should remain active during this time to insure proper microprocessor operation. RESET is active HIGH. RESET is asynchronous but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
INTR	I	The *maskable interrupt* indicates that an external interrupt has been generated. If the internal interrupt flag is set in EFLAGS, active interrupt processing will be initiated. The Intel486 Microprocessor will generate two locked interrupt acknowledge bus cycles in response to the INTR pin going active. INTR must remain active until the interrupt acknowledges have been performed to assure that the interrupt is recognized. INTR is active HIGH and is not provided with an internal pulldown resistor. INTR is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
NMI	I	The *non-maskable interrupt* request signal indicates that an external non-maskable interrupt has been generated. NMI is rising edge sensitive. NMI must be held LOW for at least four CLK periods before this rising edge. NMI is not provided with an internal pulldown resistor. NMI is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
BUS ARBITRATION		
BREQ	O	The *internal cycle pending* signal indicates that the Intel486 Microprocessor has internally generated a bus request. BREQ is generated whether or not the Intel486 Microprocessor is driving the bus. BREQ is active HIGH and is never floated.
HOLD	I	The *bus hold request* allows another bus master complete control of the Intel486 Microprocessor bus. In response to HOLD going active the Intel486 Microprocessor will float most of its output and input/output pins. HLDA will be asserted after completing the current bus cycle, burst cycle or sequence of locked cycles. The Intel486 Microprocessor will remain in this state until HOLD is deasserted. HOLD is active high and is not provided with an internal pulldown resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper operation.
HLDA	O	*Hold acknowledge* goes active in response to a hold request presented on the HOLD pin. HLDA indicates that the Intel486 microprocessor has given the bus to another local bus master. HLDA is driven active in the same clock that the Intel486 Microprocessor floats its bus. HLDA is driven inactive when leaving bus hold. HLDA is active HIGH and remains driven during bus hold.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
BUS ARBITRATION (Continued)		
BOFF#	I	The *backoff* input forces the Intel486 Microprocessor to float its bus in the next clock. The microprocessor will float all pins normally floated during bus hold but HLDA will not be asserted in response to BOFF#. BOFF# has higher priority than RDY# or BRDY#; if both are returned in the same clock, BOFF# takes effect. The microprocessor remains in bus hold until BOFF# is negated. If a bus cycle was in progress when BOFF# was asserted the cycle will be restarted. BOFF# is active LOW and must meet setup and hold times t_{18} and t_{19} for proper operation.
CACHE INVALIDATION		
AHOLD	I	The *address hold* request allows another bus master access to the Intel486 Microprocessor's address bus for a cache invalidation cycle. The Intel486 Microprocessor will stop driving its address bus in the clock following AHOLD going active. Only the address bus will be floated during address hold, the remainder of the bus will remain active. AHOLD is active HIGH and is provided with a small internal pulldown resistor. For proper operation AHOLD must meet setup and hold times t_{18} and t_{19}.
EADS#	I	This signal indicates that a *valid external address* has been driven onto the Intel486 Microprocessor address pins. This address will be used to perform an internal cache invalidation cycle. EADS# is active LOW and is provided with an internal pullup resistor. EADS# must satisfy setup and hold times t_{12} and t_{13} for proper operation.
CACHE CONTROL		
KEN#	I	The *cache enable* pin is used to determine whether the current cycle is cacheable. When the Intel486 microprocessor generates a cycle that can be cached and KEN# is active one clock before RDY# or BRDY# during the first transfer of the cycle, the cycle will become a cache line fill cycle. Returning KEN# active one clock before RDY# during the last read in the cache line fill will cause the line to be placed in the on-chip cache. KEN# is active LOW and is provided with a small internal pullup resistor. KEN# must satisfy setup and hold times t_{14} and t_{15} for proper operation.
FLUSH#	I	The *cache flush* input forces the Intel486 Microprocessor to flush its entire internal cache. FLUSH# is active low and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition in any specific clock. FLUSH# being sampled low in the clock before the falling edge of RESET causes the Intel486 Microprocessor to enter the tri-state test mode.
PAGE CACHEABILITY		
PWT PCD	O O	The *page write-through* and *page cache disable* pins reflect the state of the page attribute bits, PWT and PCD, in the page table entry or page directory entry. If paging is disabled or for cycles that are not paged, PWT and PCD reflect the state of the PWT and PCD bits in control register 3. PWT and PCD have the same timing as the cycle definition pins (M/IO#, D/C# and W/R#). PWT and PCD are active HIGH and are not driven during bus hold. PCD is masked by the cache disable bit (CD) in Control Register 0.
NUMERIC ERROR REPORTING		
FERR#	O	The *floating point error* pin is driven active when a floating point error occurs. FERR# is similar to the ERROR# pin on the 387 math coprocessor. FERR# is included for compatibility with systems using DOS type floating point error reporting. FERR# will not go active if FP errors are masked in FPU register. FERR# is active LOW, and is not floated during bus hold.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
NUMERIC ERROR REPORTING (Continued)		
IGNNE#	I	When the *ignore numeric error* pin is asserted the Intel486 Microprocessor will ignore a numeric error and continue executing non-control floating point instructions, but FERR# will still be activated by the Intel486. When IGNNE# is deasserted the Intel486 microprocessor will freeze on a non-control floating point instruction, if a previous floating point instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set. IGNNE# is active LOW and is provided with a small internal pullup resistor. IGNNE# is asynchronous but setup and hold times t_{20} and t_{21} must be met to insure recognition on any specific clock.
BUS SIZE CONTROL		
BS16# BS8#	I I	The *bus size 16* and *bus size 8* pins (bus sizing pins) cause the Intel486 Microprocessor to run multiple bus cycles to complete a request from devices that cannot provide or accept 32 bits of data in a single cycle. The bus sizing pins are sampled every clock. The state of these pins in the clock before ready is used by the Intel486 microprocessor to determine the bus size. These signals are active LOW and are provided with internal pullup resistors. These inputs must satisfy setup and hold times t_{14} and t_{15} for proper operation.
ADDRESS MASK		
A20M#	I	When the *address bit 20 mask* pin is asserted, the Intel486 Microprocessor masks physical address bit 20 (A20) before performing a lookup to the internal cache or driving a memory cycle on the bus. A20M# emulates the address wraparound at one Mbyte which occurs on the 8086. A20M# is active LOW and should be asserted only when the processor is in real mode. This pin is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For proper operation, A20M# should be sampled high at the falling edge of RESET.
TEST ACCESS PORT (50 MHz Version Only)		
TCK	I	*Test Clock* is an input to the Intel486 CPU and provides the clocking function required by the JTAG boundary scan feature. TCK is used to clock state information and data into and out of the component. State select information and data are clocked into the component on the rising edge of TCK on TMS and TDI, respectively. Data is clocked out of the part on the falling edge of TCK on TDO.
TDI	I	*Test Data Input* is the serial input used to shift JTAG instructions and data into the component. TDI is sampled on the rising edge of TCK, during the SHIFT-IR and the SHIFT-DR TAP controller states. During all other tap controller states, TDI is a "don't care".
TDO	O	*Test Data Output* is the serial output used to shift JTAG instructions and data out of the component. TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT-DR TAP controller states. At all other times TDO is driven to the high impedance state.
TMS	I	*Test Mode Select* is decoded by the JTAG TAP (Tap Access Port) to select the operation of the test logic. TMS is sampled on the rising edge of TCK. To guarantee deterministic behavior of the TAP controller TMS is provided with an internal pull-up resistor.

Table 1.1. Output Pins

Name	Active Level	When Floated
BREQ	HIGH	
HLDA	HIGH	
BE0#–BE3#	LOW	Bus Hold
PWT, PCD	HIGH	Bus Hold
W/R#, D/C#, M/IO#	HIGH	Bus Hold
LOCK#	LOW	Bus Hold
PLOCK#	LOW	Bus Hold
ADS#	LOW	Bus Hold
BLAST#	LOW	Bus Hold
PCHK#	LOW	
FERR#	LOW	
A2–A3	HIGH	Bus, Address Hold

Table 1.2. Input Pins

Name	Active Level	Synchronous/ Asynchronous
CLK		
RESET	HIGH	Asynchronous
HOLD	HIGH	Synchronous
AHOLD	HIGH	Synchronous
EADS#	LOW	Synchronous
BOFF#	LOW	Synchronous
FLUSH#	LOW	Asynchronous
A20M#	LOW	Asynchronous
BS16#, BS8#	LOW	Synchronous
KEN#	LOW	Synchronous
RDY#	LOW	Synchronous
BRDY#	LOW	Synchronous
INTR	HIGH	Asynchronous
NMI	HIGH	Asynchronous
IGNNE#	LOW	Asynchronous

Table 1.3. Input/Output Pins

Name	Active Level	When Floated
D0–D31	HIGH	Bus Hold
DP0–DP3	HIGH	Bus Hold
A4–A31	HIGH	Bus, Address Hold

Table 1.4. Test Pins (50 MHz Version Only)

Name	Input or Output	Sampled/ Driven On
TCK	Input	N/A
TDI	Input	Rising Edge of TCK
TDO	Output	Falling Edge of TCK
TMS	Input	Rising Edge of TCK

Table 1.5. Component and Revision ID

Intel486™ CPU Stepping Name	Component ID	Revision ID
B3	04	01
B4	04	01
B5	04	01
B6	04	01
C0	04	02
C1	04	03
D0	04	04
cA2	04	10
cA3	04	10
cB0	04	11
cB1	04	11
Intel OverDrive™ Processor Stepping Name		
A2	04	32
B1	04	33

2.0 ARCHITECTURAL OVERVIEW

The Intel486 Microprocessor is a 32-bit architecture with on-chip memory management, floating point and cache memory units.

The Intel486 Microprocessor contains all the features of the 386 Microprocessor with enhancements to increase performance. The instruction set includes the complete 386 microprocessor instruction set along with extensions to serve new applications. The on-chip memory management unit (MMU) is completely compatible with the 386 Microprocessor MMU. The Intel486 Microprocessor brings the 387 math coprocessor on-chip. All software written for the 386 microprocessor, 387 math coprocessor and previous members of the 86/87 architectural family will run on the Intel486 Microprocessor without any modifications.

Several enhancements have been added to the Intel486 Microprocessor to increase performance. On-chip cache memory allows frequently used data and code to be stored on-chip reducing accesses to the external bus. RISC design techniques have been used to reduce instruction cycle times. A burst bus feature enables fast cache fills. All of these features, combined, lead to performance greater than twice that of a 386 Microprocessor.

The memory management unit (MMU) consists of a segmentation unit and a paging unit. Segmentation allows management of the logical address space by providing easy data and code relocatibility and efficient sharing of global resources. The paging mechanism operates beneath segmentation and is transparent to the segmentation process. Paging is optional and can be disabled by system software. Each segment can be divided into one or more 4 Kbyte segments. To implement a virtual memory system, the Intel486 Microprocessor supports full restartability for all page and segment faults.

Memory is organized into one or more variable length segments, each up to four gigabytes (2^{32} bytes) in size. A segment can have attributes associated with it which include its location, size, type (i.e., stack, code or data), and protection characteristics. Each task on an Intel486 Microprocessor can have a maximum of 16,381 segments, each up to four gigabytes in size. Thus each task has a maximum of 64 terabytes (trillion bytes) of virtual memory.

The segmentation unit provides four-levels of protection for isolating and protecting applications and the operating system from each other. The hardware enforced protection allows the design of systems with a high degree of integrity.

The Intel486 Microprocessor has two modes of operation: Real Address Mode (Real Mode) and Protected Mode Virtual Address Mode (Protected Mode). In Real Mode the Intel486 Microprocessor operates as a very fast 8086. Real Mode is required primarily to set up the processor for Protected Mode operation. Protected Mode provides access to the sophisticated memory management paging and privilege capabilities of the processor.

Within Protected Mode, software can perform a task switch to enter into tasks designated as Virtual 8086 Mode tasks. Each virtual 8086 task behaves with 8086 semantics, allowing 8086 software (an application program or an entire operating system) to execute.

The on-chip floating point unit operates in parallel with the arithmetic and logic unit and provides arithmetic instructions for a variety of numeric data types. It executes numerous built-in transcendental functions (e.g., tangent, sine, cosine, and log functions). The floating point unit fully conforms to the ANSI/IEEE standard 754-1985 for floating point arithmetic.

The on-chip cache is 8 Kbytes in size. It is 4-way set associative and follows a write-through policy. The on-chip cache includes features to provide flexibility in external memory system design. Individual pages can be designated as cacheable or non-cacheable by software or hardware. The cache can also be enabled and disabled by software or hardware.

Finally the Intel486 Microprocessor has features to facilitate high performance hardware designs. The 1X clock eases high frequency board level designs. The burst bus feature enables fast cache fills. These features are described beginning in Section 6.

2.1 Register Set

The Intel486 Microprocessor register set includes all the registers contained in the 386 Microprocessor and the 387 math coprocessor. The register set can be split into the following categories:

- Base Architecture Registers
  - General Purpose Registers
  - Instruction Pointer
  - Flags Register
  - Segment Registers
- Systems Level Registers
  - Control Registers
  - System Address Registers

Floating Point Registers
- Data Registers
- Tag Word
- Status Word
- Instruction and Data Pointers
- Control Word

Debug and Test Registers

The base architecture and floating point registers are accessible by the applications program. The system level registers are only accessible at privilege level 0 and are used by the systems level program. The debug and test registers are also only accessible at privilege level 0.

2.1.1 BASE ARCHITECTURE REGISTERS

Figure 2.1 shows the Intel486 Microprocessor base architecture registers. The contents of these registers are task-specific and are automatically loaded with a new context upon a task switch operation.

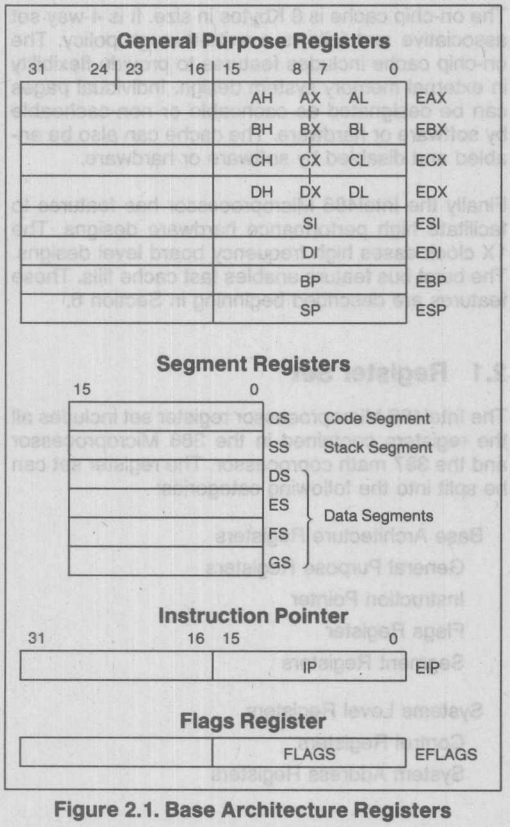


Figure 2.1. Base Architecture Registers

The base architecture includes six directly accessible descriptors, each specifying a segment up to 4 Gbytes in size. The descriptors are indicated by the selector values placed in the Intel486 Microprocessor segment registers. Various selector values can be loaded as a program executes.

The selectors are also task-specific, so the segment registers are automatically loaded with new context upon a task switch operation.

2.1.1.1 General Purpose Registers

The eight 32-bit general purpose registers are shown in Figure 2.1. These registers hold data or address quantities. The general purpose registers can support data operands of 1, 8, 16 and 32 bits, and bit fields of 1 to 32 bits. Address operands of 16 and 32 bits are supported. The 32-bit registers are named EAX, EBX, ECX, EDX, ESI, EDI, EBP and ESP.

The least significant 16 bits of the general purpose registers can be accessed separately by using the 16-bit names of the registers AX, BX, CX, DX, SI, DI, BP and SP. The upper 16 bits of the register are not changed when the lower 16 bits are accessed separately.

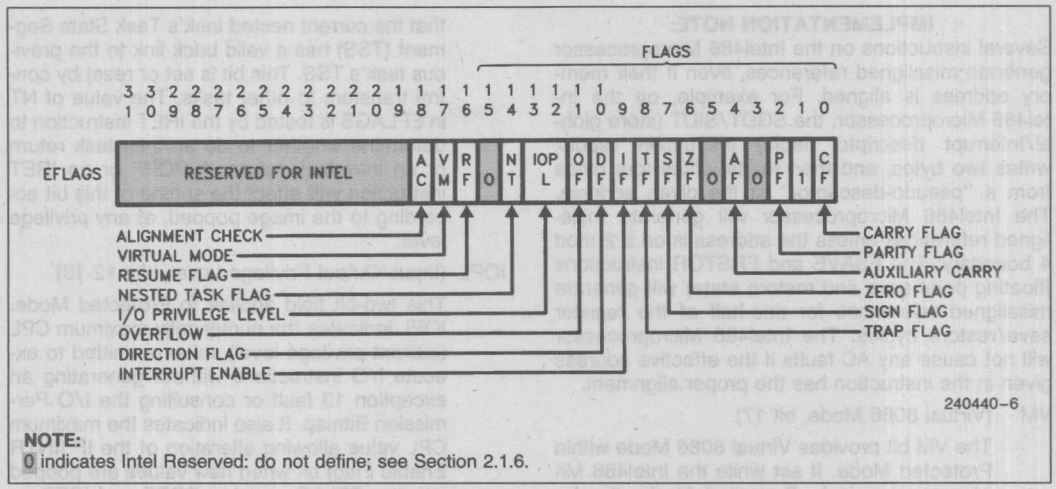
Finally 8-bit operations can individually access the lowest byte (bits 0–7) and the higher byte (bits 8–15) of the general purpose registers AX, BX, CX and DX. The lowest bytes are named AL, BL, CL and DL respectively. The higher bytes are named AH, BH, CH and DH respectively. The individual byte accessibility offers additional flexibility for data operations but is not used for effective address calculation.

2.1.1.2 Instruction Pointer

The instruction pointer, shown in Figure 2.1, is a 32-bit register named EIP. EIP holds the offset of the next instruction to be executed. The offset is always relative to the base of the code segment (CS). The lower 16 bits (bits 0–15) of the EIP contain the 16-bit instruction pointer named IP, which is used for 16-bit addressing.

2.1.1.3 Flags Register

The flags register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS control certain operations and indicate status of the Intel486 Microprocessor. The lower 16 bits (bits 0–15) of EFLAGS contain the 16-bit register named FLAGS, which is most useful when executing 8086 and 80286 code. EFLAGS is shown in Figure 2.2.



NOTE:
0 indicates Intel Reserved: do not define; see Section 2.1.6.

Figure 2.2. Flags Register

EFLAGS bits 1, 3, 5, 15 and 19–31 are "undefined". When these bits are stored during interrupt processing or with a PUSHF instruction (push flags onto stack), a one is stored in bit 1 and zeros in bits 3, 5, 15 and 19–31.

The EFLAGS register in the Intel486 Microprocessor contains a new bit not previously defined. The new bit, AC, is defined in the upper 16 bits of the register and it enables faults on accesses to misaligned data.

AC (Alignment Check, bit 18)

The AC bit enables the generation of faults if a memory reference is to a misaligned address. Alignment faults are enabled when AC is set to 1. A mis-aligned address is a word access to an odd address, a dword access to an address that is not on a dword boundary, or an 8-byte reference to an address that is not on a 64-bit word boundary. See Section 7.1.6 for more information on operand alignment.

Alignment faults are only generated by programs running at privilege level 3. The AC bit setting is ignored at privilege levels 0, 1 and 2. Note that references to the descriptor tables (for selector loads), or the task state segment (TSS), are implicitly level 0 references even if the instructions causing the references are executed at level 3. Alignment faults are reported through interrupt 17, with an error code of 0. Table 2.1 gives the alignment required for the Intel486 microprocessor data types.

Table 2.1. Data Type Alignment Requirements

Memory Access	Alignment (Byte Boundary)
Word	2
Dword	4
Single Precision Real	4
Double Precision Real	8
Extended Precision Real	8
Selector	2
48-Bit Segmented Pointer	4
32-Bit Flat Pointer	4
32-Bit Segmented Pointer	2
48-Bit "Pseudo-Descriptor"	4
FSTENV/FLDENV Save Area	4/2 (On Operand Size)
FSAVE/FRSTOR Save Area	4/2 (On Operand Size)
Bit String	4

IMPLEMENTATION NOTE:
Several instructions on the Intel486 Microprocessor generate misaligned references, even if their memory address is aligned. For example, on the Intel486 Microprocessor, the SGDT/SIDT (store global/interrupt descriptor table) instruction reads/writes two bytes, and then reads/writes four bytes from a "pseudo-descriptor" at the given address. The Intel486 Microprocessor will generate misaligned references unless the address is on a 2 mod 4 boundary. The FSAVE and FRSTOR instructions (floating point save and restore state) will generate misaligned references for one-half of the register save/restore cycles. The Intel486 Microprocessor will not cause any AC faults if the effective address given in the instruction has the proper alignment.

VM (Virtual 8086 Mode, bit 17)

The VM bit provides Virtual 8086 Mode within Protected Mode. If set while the Intel486 Microprocessor is in Protected Mode, the Intel486 Microprocessor will switch to Virtual 8086 operation, handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes. The VM bit can be set only in Protected Mode, by the IRET instruction (if current privilege level = 0) and by task switches at any privilege level. The VM bit is unaffected by POPF. PUSHF always pushes a 0 in this bit, even if executing in Virtual 8086 Mode. The EFLAGS image pushed during interrupt processing or saved during task switches will contain a 1 in this bit if the interrupted code was executing as a Virtual 8086 Task.

RF (Resume Flag, bit 16)

The RF flag is used in conjunction with the debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. When RF is set, it causes any debug fault to be ignored on the next instruction. RF is then automatically reset at the successful completion of every instruction (no faults are signalled) except the IRET instruction, the POPF instruction, (and JMP, CALL, and INT instructions causing a task switch). These instructions set RF to the value specified by the memory image. For example, at the end of the breakpoint service routine, the IRET instruction can pop an EFLAG image having the RF bit set and resume the program's execution at the breakpoint address without generating another breakpoint fault on the same location.

NT (Nested Task, bit 14)

This flag applies to Protected Mode. NT is set to indicate that the execution of this task is nested within another task. If set, it indicates that the current nested task's Task State Segment (TSS) has a valid back link to the previous task's TSS. This bit is set or reset by control transfers to other tasks. The value of NT in EFLAGS is tested by the IRET instruction to determine whether to do an inter-task return or an intra-task return. A POPF or an IRET instruction **will** affect the setting of this bit according to the image popped, at any privilege level.

IOPL (Input/Output Privilege Level, bits 12-13)

This two-bit field applies to Protected Mode. IOPL indicates the numerically maximum CPL (current privilege level) value permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O Permission Bitmap. It also indicates the maximum CPL value allowing alteration of the IF (INTR Enable Flag) bit when new values are popped into the EFLAG register. POPF and IRET instruction can alter the IOPL field when executed at CPL = 0. Task switches can always alter the IOPL field, when the new flag image is loaded from the incoming task's TSS.

OF (Overflow Flag, bit 11)

OF is set if the operation resulted in a signed overflow. Signed overflow occurs when the operation resulted in carry/borrow **into** the sign bit (high-order bit) of the result but did not result in a carry/borrow **out of** the high-order bit, or vice-versa. For 8-, 16-, 32-bit operations, OF is set according to overflow at bit 7, 15, 31, respectively.

DF (Direction Flag, bit 10)

DF defines whether ESI and/or EDI registers postdecrement or postincrement during the string instructions. Postincrement occurs if DF is reset. Postdecrement occurs if DF is set.

IF (INTR Enable Flag, bit 9)

The IF flag, when set, allows recognition of external interrupts signalled on the INTR pin. When IF is reset, external interrupts signalled on the INTR are not recognized. IOPL indicates the maximum CPL value allowing alteration of the IF bit when new values are popped into EFLAGS or FLAGS.

TF (Trap Enable Flag, bit 8)

TF controls the generation of exception 1 trap when single-stepping through code. When TF is set, the Intel486 Microprocessor generates an exception 1 trap after the next instruction is executed. When TF is reset, exception 1 traps occur only as a function of the breakpoint addresses loaded into debug registers DR0–DR3.

SF (Sign Flag, bit 7)

SF is set if the high-order bit of the result is set, it is reset otherwise. For 8-, 16-, 32-bit operations, SF reflects the state of bit 7, 15, 31 respectively.

ZF (Zero Flag, bit 6)

ZF is set if all bits of the result are 0. Otherwise it is reset.

AF (Auxiliary Carry Flag, bit 4)

The Auxiliary Flag is used to simplify the addition and subtraction of packed BCD quantities. AF is set if the operation resulted in a carry out of bit 3 (addition) or a borrow into bit 3 (subtraction). Otherwise AF is reset. AF is affected by carry out of, or borrow into bit 3 only, regardless of overall operand length: 8, 16 or 32 bits.

PF (Parity Flags, bit 2)

PF is set if the low-order eight bits of the operation contains an even number of "1's" (even parity). PF is reset if the low-order eight bits have odd parity. PF is a function of only the low-order eight bits, regardless of operand size.

CF (Carry Flag, bit 0)

CF is set if the operation resulted in a carry out of (addition), or a borrow into (subtraction) the high-order bit. Otherwise CF is reset. For 8-, 16- or 32-bit operations, CF is set according to carry/borrow at bit 7, 15 or 31, respectively.

NOTE:
In these descriptions, "set" means "set to 1," and "reset" means "reset to 0."

2.1.1.4 Segment Registers

Six 16-bit segment registers hold segment selector values identifying the currently addressable memory segments. In protected mode, each segment may range in size from one byte up to the entire linear and physical address space of the machine, 4 Gbytes (2^{32} bytes). In real address mode, the maximum segment size is fixed at 64 Kbytes (2^{16} bytes).

The six addressable segments are defined by the segment registers CS, SS, DS, ES, FS and GS. The selector in CS indicates the current code segment; the selector in SS indicates the current stack segment; the selectors in DS, ES, FS and GS indicate the current data segments.

2.1.1.5 Segment Descriptor Cache Registers

The segment descriptor cache registers are not programmer visible, yet it is very useful to understand their content. A programmer invisible descriptor cache register is associated with each programmer-visible segment register, as shown by Figure 2.3. Each descriptor cache register holds a 32-bit base address, a 32-bit segment limit, and the other necessary segment attributes.

SEGMENT REGISTERS		DESCRIPTOR REGISTERS (LOADED AUTOMATICALLY)			
15 0		Physical Base Address	Segment Limit	Other Segment Attributes from Descriptor	
Selector	CS–				—
Selector	SS–			—	—
Selector	DS–			—	—
Selector	ES–			—	—
Selector	FS–			—	—
Selector	GS–			—	—

Figure 2.3. Intel486™ Microprocessor Segment Registers and Associated Descriptor Cache Registers

When a selector value is loaded into a segment register, the associated descriptor cache register is automatically updated with the correct information. In Real Address Mode, only the base address is updated directly (by shifting the selector value four bits to the left), since the segment maximum limit and attributes are fixed in Real Mode. In Protected Mode, the base address, the limit, and the attributes are all updated per the contents of the segment descriptor indexed by the selector.

Whenever a memory reference occurs, the segment descriptor cache register associated with the segment being used is automatically involved with the memory reference. The 32-bit segment base address becomes a component of the linear address calculation, the 32-bit limit is used for the limit-check operation, and the attributes are checked against the type of memory reference requested.

2.1.2 SYSTEM LEVEL REGISTERS

The system level registers, Figure 2.4, control operation of the on-chip cache, the on-chip floating point unit (FPU) and the segmentation and paging mechanisms. These registers are only accessible to programs running at privilege level 0, the highest privilege level.

The system level registers include three control registers and four segmentation base registers. The three control registers are CR0, CR2 and CR3. CR1 is reserved for future Intel processors. The four segmentation base registers are the Global Descriptor Table Register (GDTR), the Interrupt Descriptor Table Register (IDTR), the Local Descriptor Table Register (LDTR) and the Task State Segment Register (TR).

2.1.2.1 Control Registers

Control Register 0 (CR0)

CR0, shown in Figure 2.5, contains 10 bits for control and status purposes. Five of the bits defined in the Intel486 Microprocessor's CR0 are newly defined. The new bits are CD, NW, AM, WP and NE. The function of the bits in CR0 can be categorized as follows:

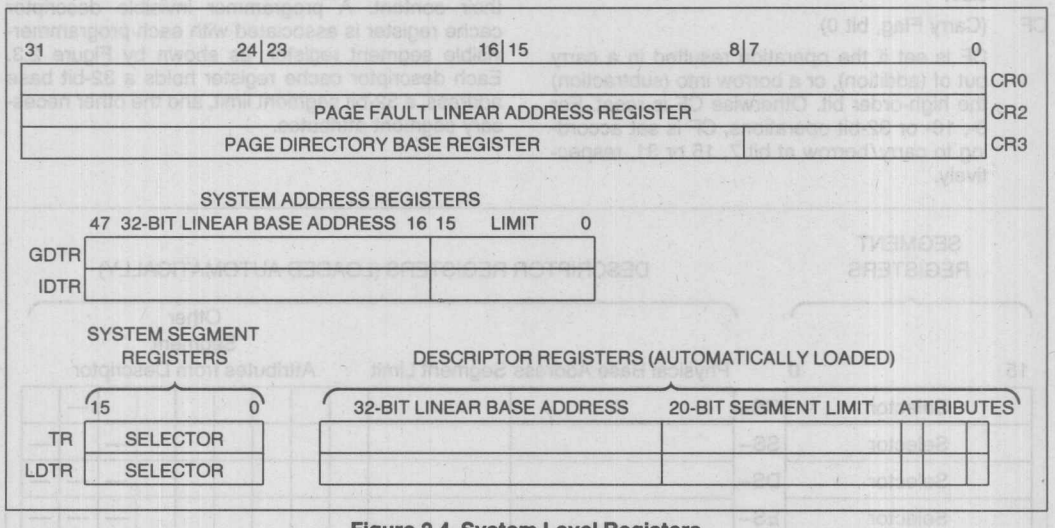
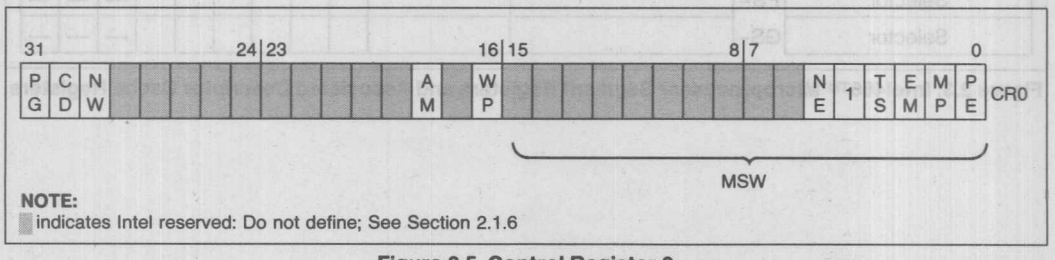


Figure 2.4. System Level Registers



NOTE:
indicates Intel reserved: Do not define; See Section 2.1.6

Figure 2.5. Control Register 0

Intel486 Microprocessor Operating Modes: PG, PE (Table 2.2)

On-Chip Cache Control Modes: CD, NW (Table 2.3)

On-Floating Point Unit Control: TS, EM, MP, NE (Table 2.4)

Alignment Check Control: AM

Supervisor Write Protect: WP

Table 2.2. Processor Operating Modes

PG	PE	Mode
0	0	REAL Mode. Exact 8086 semantics, with 32-bit extensions available with prefixes.
0	1	Protected Mode. Exact 80286 semantics, plus 32-bit extensions through both prefixes and "default" prefix setting associated with code segment descriptors. Also, a sub-mode is defined to support a virtual 8086 within the context of the extended 80286 protection model.
1	0	UNDEFINED. Loading CR0 with this combination of PG and PE bits will raise a GP fault with error code 0.
1	1	Paged Protected Mode. All the facilities of Protected mode, with paging enabled underneath segmentation.

Table 2.3. On-Chip Cache Control Modes

CD	NW	Operating Mode
1	1	Cache fills disabled, write-through and invalidates disabled.
1	0	Cache fills disabled, write-through and invalidates enabled.
0	1	INVALID. If CR0 is loaded with this configuration of bits, a GP fault with error code is raised.
0	0	Cache fills enabled, write-through and invalidates enabled.

Table 2.4. On-Chip Floating Point Unit Control

CR0 BIT			Instruction Type	
EM	TS	MP	Floating-Point	Wait
0	0	0	Execute	Execute
0	0	1	Execute	Execute
0	1	0	Trap 7	Execute
0	1	1	Trap 7	Trap 7
1	0	0	Trap 7	Execute
1	0	1	Trap 7	Execute
1	1	0	Trap 7	Execute
1	1	1	Trap 7	Trap 7

The low-order 16 bits of CR0 are also known as the Machine Status Word (MSW), for compatibility with the 80286 protected mode. LMSW and SMSW (load and store MSW) instructions are taken as special aliases of the load and store CR0 operations, where only the low-order 16 bits of CR0 are involved. The LMSW and SMSW instructions in the Intel486 microprocessor work in an identical fashion to the LMSW and SMSW instructions in the 80286 (i.e., they only operate on the low-order 16 bits of CR0 and ignores the new bits). New Intel486 Microprocessor operating systems should use the MOV CR0, Reg instruction.

The defined CR0 bits are described below.

PG (Paging Enable, bit 31)

The PG bit is used to indicate whether paging is enabled (PG = 1) or disabled (PG = 0). See Table 2.2.

CD (Cache Disable, bit 30)

The CD bit is used to enable the on-chip cache. When CD = 1, the cache will not be filled on cache misses. When CD = 0, cache fills may be performed on misses. See Table 2.3.

The state of the CD bit, the cache enable input pin (KEN#), and the relevant page cache disable (PCD) bit determine if a line read in response to a cache miss will be installed in the cache. A line is installed in the cache only if CD = 0 and KEN# and PCD are both zero. The relevant PCD bit comes from either the page table entry, page directory entry or control register 3. Refer to Section 5.6 for more details on page cacheability.

CD is set to one after RESET.

NW (Not Write-Through, bit 29)

The NW bit enables on-chip cache write-throughs and write-invalidate cycles (NW = 0). When NW = 0, all writes, including cache hits, are sent out to the pins. Invalidate cycles are enabled when NW = 0. During an invalidate cycle a line will be removed from the cache if the invalidate address hits in the cache. See Table 2.3.

When NW = 1, write-throughs and write-invalidate cycles are disabled. A write will not be sent to the pins if the write hits in the cache. With NW = 1 the only write cycles that reach the external bus are cache misses. Write hits with NW = 1 will never update main memory. Invalidate cycles are ignored when NW = 1.

AM (Alignment Mask, bit 18)

The AM bit controls whether the alignment check (AC) bit in the flag register (EFLAGS) can allow an alignment fault. AM = 0 disables the AC bit. AM = 1 enables the AC bit. AM = 0 is the 386 Microprocessor compatible mode.

386 Microprocessor software may load incorrect data into the AC bit in the EFLAGS register. Setting AM=0 will prevent AC faults from occurring before the Intel486 Microprocessor has created the AC interrupt service routine.

WP (Write Protect, bit 16)

WP protects read-only pages from supervisor write access. The 386 Microprocessor allows a read-only page to be written from privilege levels 0–2. The Intel486 Microprocessor is compatible with the 386 Microprocessor when WP=0. WP=1 forces a fault on a write to a read-only page from any privilege level. Operating systems with Copy-on-Write features can be supported with the WP bit. Refer to Section 4.5.3 for further details on use of the WP bit.

NE (Numerics Exception, bit 5)

The NE bit controls whether unmasked floating point exceptions (UFPE) are handled through interrupt vector 16 (NE=1) or through an external interrupt (NE=0). NE=0 (default at reset) supports the DOS operating system error reporting scheme from the 8087, 80287 and 387 math coprocessor. In DOS systems, math coprocessor errors are reported via external interrupt vector 13. DOS uses interrupt vector 16 for an operating system call. Refer to Sections 6.2.13 and 7.2.14 for more information on floating point error reporting.

For any UFPE the floating point error output pin (FERR#) will be driven active.

For NE=0, the Intel486 Microprocessor works in conjunction with the ignore numeric error input (IGNNE#) and the FERR# output pins. When a UFPE occurs and the IGNNE# input is inactive, the Intel486 Microprocessor freezes immediately before executing the next floating point instruction. An external interrupt controller will supply an interrupt vector when FERR# is driven active. The UFPE is ignored if IGNNE# is active and floating point execution continues.

NOTE:

The freeze does not take place if the next instruction is one of the control instructions FNCLEX, FNINIT, FNSAVE, FNSTENV, FNSTCW, FNSTSW, FNSTSW AX, FNENI, FNDISI and FNSETPM. The freeze does occur if the next instruction is WAIT.

For NE=1, any UFPE will result in a software interrupt 16, immediately before executing the next non-control floating point or WAIT instruction. The ignore numeric error input (IGNNE#) signal will be ignored.

TS (Task Switched, bit 3)

The TS bit is set whenever a task switch operation is performed. Execution of a floating point instruction with TS=1 will cause a device not available (DNA) fault (trap vector 7). If TS=1 and MP=1 (monitor coprocessor in CR0) a WAIT instruction will cause a DNA fault. See Table 2.4.

EM (Emulate Coprocessor, bit 2)

The EM bit determines whether floating point instructions are trapped (EM=1) or executed. If EM=1, all floating point instructions will cause fault 7.

NOTE:

WAIT instructions are not affected by the state of EM. See Table 2.4.

MP (Monitor Coprocessor, bit 1)

The MP bit is used in conjunction with the TS bit to determine if WAIT instructions should trap. If MP=1 and TS=1, WAIT instructions cause fault 7. Refer to Table 2.4. The TS bit is set to 1 on task switches by the Intel486 Microprocessor. Floating point instructions are not affected by the state of the MP bit. It is recommended that the MP bit be set to one for the normal operation of the Intel486 Microprocessor.

PE (Protection Enable, bit 0)

The PE bit enables the segment based protection mechanism. If PE=1 protection is enabled. When PE=0 the Intel486 Microprocessor operates in REAL mode, with segment based protection disabled, and addresses formed as in an 8086. Refer to Table 2.2.

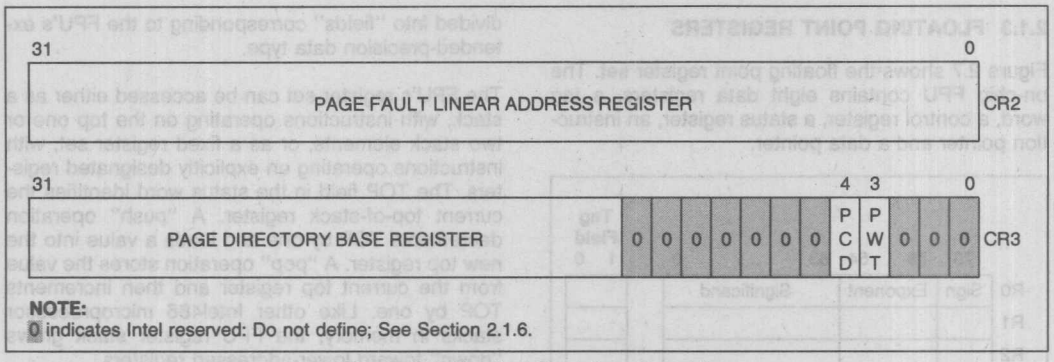
All new CR0 bits added to the 386 and Intel486 Microprocessors, except for ET and NE, are upward compatible with the 80286 because they are in register bits not defined in the 80286. For strict compatibility with the 80286, the load machine status word (LMSW) instruction is defined to not change the ET or NE bits.

Control Register 1 (CR1)

CR1 is reserved for use in future Intel microprocessors.

Control Register 2 (CR2)

CR2, shown in Figure 2.6, holds the 32-bit linear address that caused the last page fault detected. The error code pushed onto the page fault handler's stack when it is invoked provides additional status information on this page fault.



NOTE:
0 indicates Intel reserved: Do not define; See Section 2.1.6.

Figure 2.6. Control Registers 2 and 3

Control Register 3 (CR3)

CR3, shown in Figure 2.6, contains the physical base address of the page directory table. The Intel486 Microprocessor page directory is always page aligned (4 Kbyte-aligned). This alignment is enforced by only storing bits 20–31 in CR3.

In the Intel486 Microprocessor CR3 contains two new bits, page write-through (PWT) (bit 3) and page cache disable (PCD) (bit 4). The page table entry (PTE) and page directory entry (PDE) also contain PWT and PCD bits. PWT and PCD control page cacheability. When a page is accessed in external memory, the state of PWT and PCD are driven out on the PWT and PCD pins. The source of PWT and PCD can be CR3, the PTE or the PDE. PWT and PCD are sourced from CR3 when the PDE is being updated. When paging is disabled (PG = 0 in CR0), PCD and PWT are assumed to be 0, regardless of their state in CR3.

A task switch through a task state segment (TSS) which changes the values in CR3, or an explicit load into CR3 with any value, will invalidate all cached page table entries in the translation lookaside buffer (TLB).

The page directory base address in CR3 is a physical address. The page directory can be paged out while its associated task is suspended, but the operating system must ensure that the page directory is resident in physical memory before the task is dispatched. The entry in the TSS for CR3 has a physical address, with no provision for a present bit. This means that the page directory for a task must be resident in physical memory. The CR3 image in a TSS must point to this area, before the task can be dispatched through its TSS.

2.1.2.2 System Address Registers

Four special registers are defined to reference the tables or segments supported by the 80286, 386 and Intel486 Microprocessor protection model. These tables or segments are:

GDT (Global Descriptor Table)
IDT (Interrupt Descriptor Table)
LDT (Local Descriptor Table)
TSS (Task State Segment)

The addresses of these tables and segments are stored in special registers, the System Address and System Segment Registers, illustrated in Figure 2.4. These registers are named GDTR, IDTR, LDTR and TR respectively. Section 4, Protected Mode Architecture, describes the use of these registers.

System Address Registers: GDTR and IDTR

The GDTR and IDTR hold the 32-bit linear base address and 16-bit limit of the GDT and IDT, respectively.

Since the GDT and IDT segments are global to all tasks in the system, the GDT and IDT are defined by 32-bit linear addresses (subject to page translation if paging is enabled) and 16-bit limit values.

System Segment Registers: LDTR and TR

The LDTR and TR hold the 16-bit selector for the LDT descriptor and the TSS descriptor, respectively.

Since the LDT and TSS segments are task specific segments, the LDT and TSS are defined by selector values stored in the system segment registers.

NOTE:
A programmer-invisible segment descriptor register is associated with each system segment register.

2.1.3 FLOATING POINT REGISTERS

Figure 2.7 shows the floating point register set. The on-chip FPU contains eight data registers, a tag word, a control register, a status register, an instruction pointer and a data pointer.

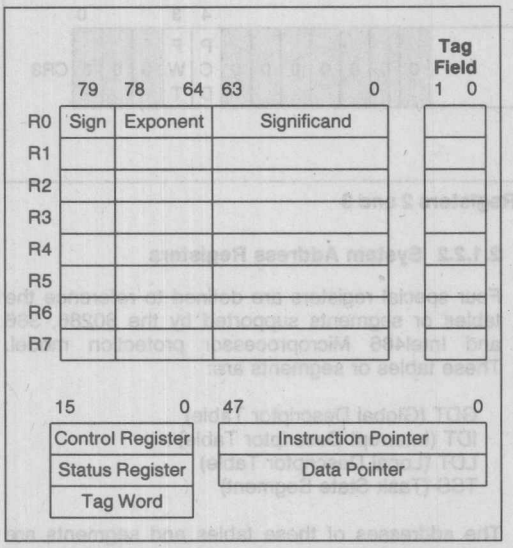


Figure 2.7. Floating Point Registers

The operation of the Intel486 Microprocessor's on-chip floating point unit is exactly the same as the 387 math coprocessor. Software written for the 387 math coprocessor will run on the on-chip floating point unit (FPU) without any modifications.

2.1.3.1 Data Registers

Floating point computations use the Intel486 Microprocessor's FPU data registers. These eight 80-bit registers provide the equivalent capacity of twenty 32-bit registers. Each of the eight data registers is divided into "fields" corresponding to the FPU's extended-precision data type.

The FPU's register set can be accessed either as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers. The TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by one and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by one. Like other Intel486 microprocessor stacks in memory, the FPU register stack grows "down" toward lower-addressed registers.

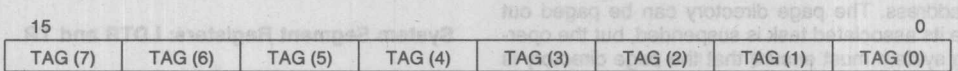
Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to use. This explicit register addressing is also relative to TOP.

2.1.3.2 Tag Word

The tag word marks the content of each numeric data register, as shown in Figure 2.8. Each two-bit tag represents one of the eight data registers. The principal function of the tag word is to optimize the FPUs performance and stack handling by making it possible to distinguish between empty and nonempty register locations. It also enables exception handlers to check the contents of a stack location without the need to perform complex decoding of the actual data.

2.1.3.3 Status Word

The 16-bit status word reflects the overall state of the FPU. The status word is shown in Figure 2.9 and is located in the status register.



NOTE:
The index i of tag(i) is **not** top-relative. A program typically uses the "top" field of Status Word to determine which tag(i) field refers to logical top of stack.

TAG VALUES:
- 00 = Valid
- 01 = Zero
- 10 = QNaN, SNaN, Infinity, Denormal and Unsupported Formats
- 11 = Empty

Figure 2.8. FPU Tag Word

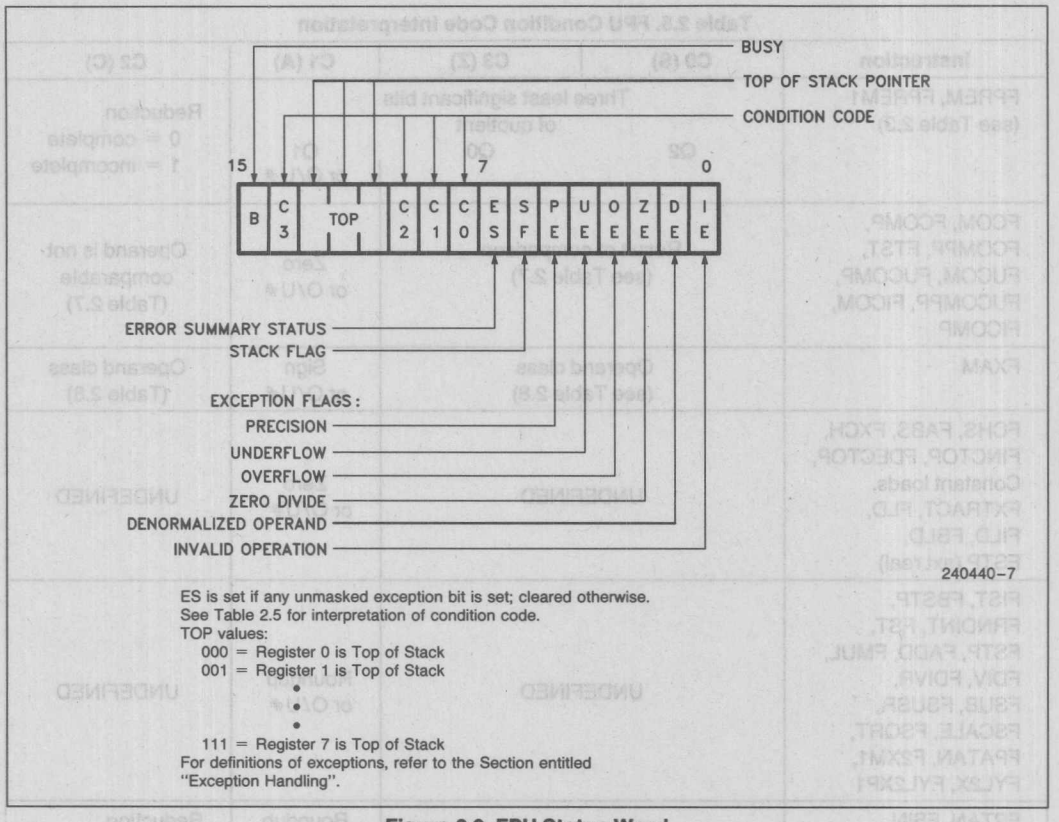


Figure 2.9. FPU Status Word

The B bit (Busy, bit 15) is included for 8087 compatibility. The B bit reflects the contents of the ES bit (bit 7 of the status word).

Bits 13–11 (TOP) point to the FPU register that is the current top-of-stack.

The four numeric condition code bits, C0–C3, are similar to the flags in EFLAGS. Instructions that perform arithmetic operations update C0–C3 to reflect the outcome. The effects of these instructions on the condition codes are summarized in Tables 2.5 through 2.8.

Table 2.5. FPU Condition Code Interpretation

Instruction	C0 (S)	C3 (Z)	C1 (A)	C2 (C)
FPREM, FPREM1 (see Table 2.3)	Three least significant bits of quotient Q2	Q0	Q1 or O/U#	Reduction 0 = complete 1 = incomplete
FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of comparison (see Table 2.7)		Zero or O/U#	Operand is not comparable (Table 2.7)
FXAM	Operand class (see Table 2.8)		Sign or O/U#	Operand class (Table 2.8)
FCHS, FABS, FXCH, FINCTOP, FDECTOP, Constant loads, FXTRACT, FLD, FILD, FBLD, FSTP (ext real)	UNDEFINED		Zero or O/U#	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/U#	UNDEFINED
FPTAN, FSIN FCOS, FSINCOS	UNDEFINED		Roundup or O/U#, undefined if C2 = 1	Reduction 0 = complete 1 = incomplete
FLDENV, FRSTOR	Each bit loaded from memory			
FINIT	Clears these bits			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FSAVE	UNDEFINED			

O/U#	When both IE and SF bits of status word are set, indicating a stack exception, this bit distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).
Reduction	If FPREM or FPREM1 produces a remainder that is less than the modulus, reduction is complete. When reduction is incomplete the value at the top of the stack is a partial remainder, which can be used as input to further reduction. For FPTAN, FSIN, FCOS, and FSINCOS, the reduction bit is set if the operand at the top of the stack is too large. In this case the original operand remains at the top of the stack.
Roundup	When the PE bit of the status word is set, this bit indicates whether the last rounding in the instruction was upward.
UNDEFINED	Do not rely on finding any specific value in these bits.

Table 2.6. Condition Code Interpretation after FPREM and FPREM1 Instructions

Condition Code				Interpretation after FPREM and FPREM1	
C2	C3	C1	C0		
1	X	X	X	Incomplete Reduction: further interaction required for complete reduction	
	Q1	Q0	Q2	Q MOD8	
0	0	0	0	0	Complete Reduction: C0, C3, C1 contain three least significant bits of quotient
	0	1	0	1	
	1	0	0	2	
	1	1	0	3	
	0	0	1	4	
	0	1	1	5	
	1	0	1	6	
	1	1	1	7	

Table 2.7. Condition Code Resulting from Comparison

Order	C3	C2	C0
TOP > Operand	0	0	0
TOP < Operand	0	0	1
TOP = Operand	1	0	0
Unordered	1	1	1

Table 2.8. Condition Code Defining Operand Class

C3	C2	C1	C0	Value at TOP
0	0	0	0	+ Unsupported
0	0	0	1	+ NaN
0	0	1	0	− Unsupported
0	0	1	1	− NaN
0	1	0	0	+ Normal
0	1	0	1	+ Infinity
0	1	1	0	− Normal
0	1	1	1	− Infinity
1	0	0	0	+ 0
1	0	0	1	+ Empty
1	0	1	0	− 0
1	0	1	1	− Empty
1	1	0	0	+ Denormal
1	1	1	0	− Denormal

Bit 7 is the error summary (ES) status bit. The ES bit is set if any unmasked exception bit (bits 0–5 in the status word) is set; ES is clear otherwise. The FERR# (floating point error) signal is asserted when ES is set.

Bit 6 is the stack flag (SF). This bit is used to distinguish invalid operations due to stack overflow or underflow. When SF is set, bit 9 (C1) distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).

Table 2.9 shows the six exception flags in bits 0–5 of the status word. Bits 0–5 are set to indicate that the FPU has detected an exception while executing an instruction.

The six exception flags in the status word can be individually masked by mask bits in the FPU control word. Table 2.9 lists the exception conditions, and their causes in order of precedence. Table 2.9 also shows the action taken by the FPU if the corresponding exception flag is masked.

An exception that is not masked by the control word will cause three things to happen: the corresponding exception flag in the status word will be set, the ES bit in the status word will be set and the FERR# output signal will be asserted. When the Intel486 Microprocessor attempts to execute another floating point or WAIT instruction, exception 16 occurs or an external interrupt happens if the NE = 1 in control register 0. The exception condition must be resolved via an interrupt service routine. The FPU saves the address of the floating point instruction that caused the exception and the address of any memory operand required by that instruction in the instruction and data pointers (see Section 2.1.3.4).

Note that when a new value is loaded into the status word by the FLDENV (load environment) or FRSTOR (restore state) instruction, the value of ES (bit 7) and its reflection in the B bit (bit 15) are not derived from the values loaded from memory. The values of ES and B are dependent upon the values of the exception flags in the status word and their corresponding masks in the control word. If ES is set in such a case, the FERR# output of the Intel486 Microprocessor is activated immediately.

2.1.3.4 Instruction and Data Pointers

Because the FPU operates in parallel with the ALU (in the Intel486 microprocessor the arithmetic and logic unit (ALU) consists of the base architecture registers), any errors detected by the FPU may be reported after the ALU has executed the floating point instruction that caused it. To allow identification of the failing numeric instruction, the Intel486 Microprocessor contains two pointer registers that supply the address of the failing numeric instruction and the address of its numeric memory operand (if appropriate).

Table 2.9. FPU Exceptions

Exception	Cause	Default Action (if exception is masked)
Invalid Operation	Operation on a signaling NaN, unsupported format, indeterminate form ($0 * \infty$, 0/0, $(+\infty) + (-\infty)$, etc.), or stack overflow/underflow (SF is also set).	Result is a quiet NaN, integer indefinite, or BCD indefinite
Denormalized Operand	At least one of the operands is denormalized, i.e., it has the smallest exponent but a nonzero significand.	Normal processing continues
Zero Divisor	The divisor is zero while the dividend is a noninfinite, nonzero number.	Result is ∞
Overflow	The result is too large in magnitude to fit in the specified format.	Result is largest finite value or ∞
Underflow	The true result is nonzero but too small to be represented in the specified format, and, if underflow exception is masked, denormalization causes loss of accuracy.	Result is denormalized or zero
Inexact Result (Precision)	The true result is not exactly representable in the specified format (e.g., 1/3); the result is rounded according to the rounding mode.	Normal processing continues

The instruction and data pointers are provided for user-written error handlers. These registers are accessed by the FLDENV (load environment), FSTENV (store environment), FSAVE (save state) and FRSTOR (restore state) instructions. Whenever the Intel486 Microprocessor decodes a new floating point instruction, it saves the instruction (including any prefixes that may be present), the address of the operand (if present) and the opcode.

The instruction and data pointers appear in one of four formats depending on the operating mode of the Intel486 Microprocessor (protected mode or real-address mode) and depending on the operand-size attribute in effect (32-bit operand or 16-bit operand). When the Intel486 Microprocessor is in the virtual-86 mode, the real address mode formats are used. The four formats are shown in Figures 2.10–2.13. The floating point instructions FLDENV, FSTENV, FSAVE and FRSTOR are used to transfer these values to and from memory. Note that the value of the data pointer is undefined if the prior floating point instruction did not have a memory operand.

NOTE:
The operand size attribute is the D bit in a segment descriptor.

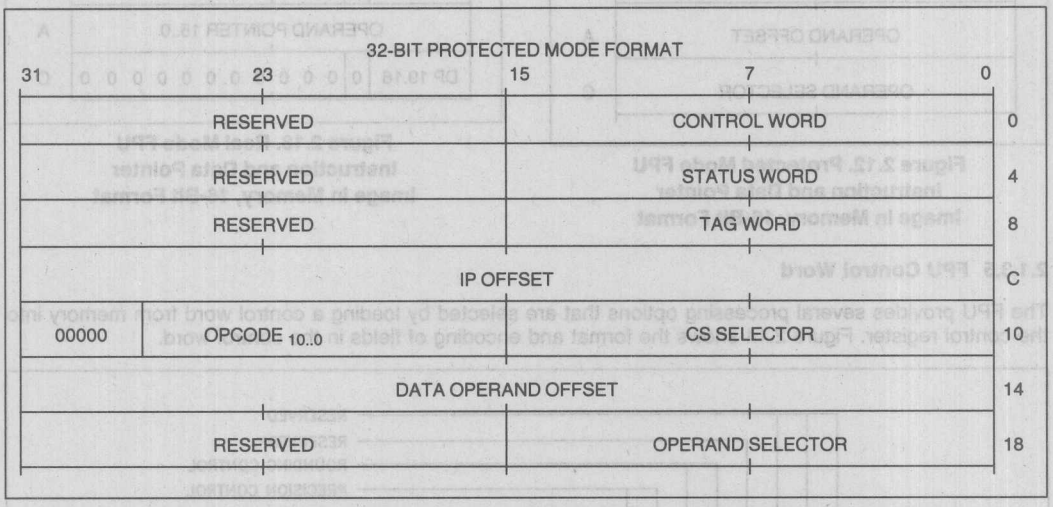


Figure 2.10. Protected Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

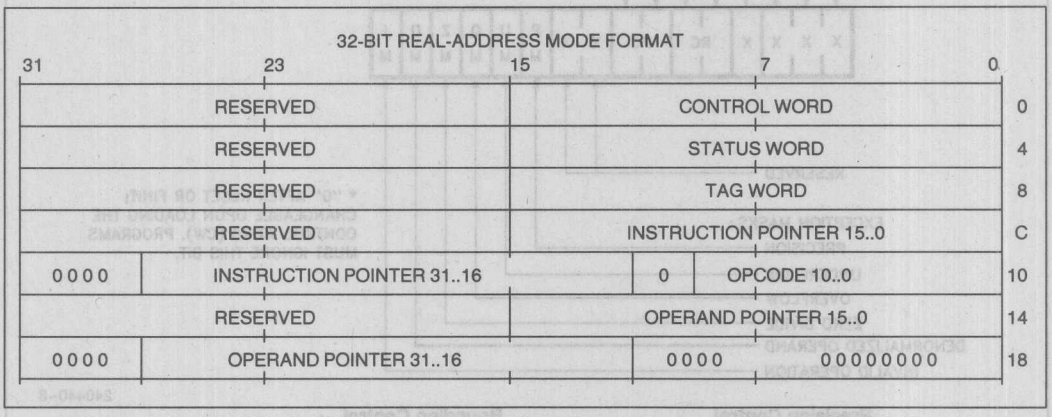


Figure 2.11. Real Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

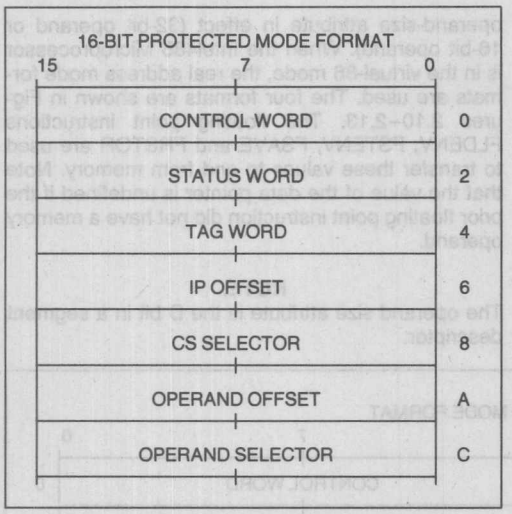


Figure 2.12. Protected Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

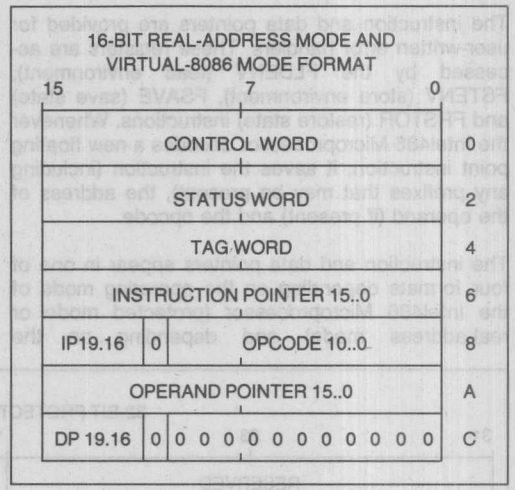


Figure 2.13. Real Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

2.1.3.5 FPU Control Word

The FPU provides several processing options that are selected by loading a control word from memory into the control register. Figure 2.14 shows the format and encoding of fields in the control word.

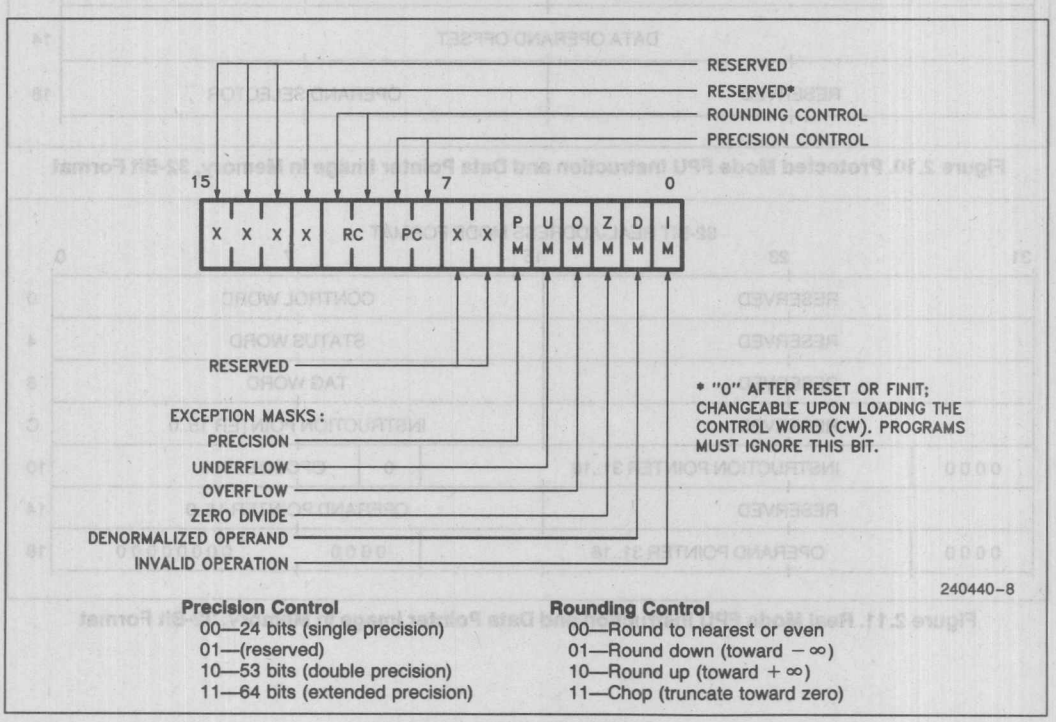


Figure 2.14. FPU Control Word

The low-order byte of the FPU control word configures the FPU error and exception masking. Bits 0–5 of the control word contain individual masks for each of the six exceptions that the FPU recognizes.

The high-order byte of the control word configures the FPU operating mode, including precision and rounding.

RC (Rounding Control, bits 10–11)

The RC bits provide for directed rounding and true chop, as well as the unbiased round to nearest even mode specified in the IEEE standard. Rounding control affects only those instructions that perform rounding at the end of the operation (and thus can generate a precision exception); namely, FST, FSTP, FIST, all arithmetic instructions (except FPREM, FPREM1, FXTRACT, FABS and FCHS), and all transcendental instructions.

PC (Precision Control, bits 8–9)

The PC bits can be used to set the FPU internal operating precision of the significand at less than the default of 64 bits (extended precision). This can be useful in providing compatibility with early generation arithmetic processors of smaller precision. PC affects only the instructions ADD, SUB, DIV, MUL, and SQRT. For all other instructions, either the precision is determined by the opcode or extended precision is used.

2.1.4 DEBUG AND TEST REGISTERS

2.1.4.1 Debug Registers

The six programmer accessible debug registers, Figure 2.15, provide on-chip support for debugging. Debug registers DR0–3 specify the four linear breakpoints. The Debug control register DR7, is used to set the breakpoints and the Debug Status Register, DR6, displays the current state of the breakpoints. The use of the Debug registers is described in Section 9.

Debug Registers

LINEAR BREAKPOINT ADDRESS 0	DR0
LINEAR BREAKPOINT ADDRESS 1	DR1
LINEAR BREAKPOINT ADDRESS 2	DR2
LINEAR BREAKPOINT ADDRESS 3	DR3
Intel Reserved Do Not Define	DR4
Intel Reserved Do Not Define	DR5
BREAKPOINT STATUS	DR6
BREAKPOINT CONTROL	DR7

Test Registers

CACHE TEST DATA	TR3
CACHE TEST STATUS	TR4
CACHE TEST CONTROL	TR5
TLB TEST CONTROL	TR6
TLB TEST STATUS	TR7

TLB = Translation Lookaside Buffer

Figure 2.15

2.1.4.2 Test Registers

The Intel486 Microprocessor contains five test registers. The test registers are shown in Figure 2.15. TR6 and TR7 are used to control the testing of the translation lookaside buffer. TR3, TR4 and TR5 are used for testing the on-chip cache. The use of the test registers is discussed in Section 8.

2.1.5 REGISTER ACCESSIBILITY

There are a few differences regarding the accessibility of the registers in Real and Protected Mode. Table 2.10 summarizes these differences. See Section 4, Protected Mode Architecture, for further details.

Table 2.10. Register Usage

Register	Use in Real Mode		Use in Protected Mode		Use in Virtual 8086 Mode	
	Load	Store	Load	Store	Load	Store
General Registers	Yes	Yes	Yes	Yes	Yes	Yes
Segment Register	Yes	Yes	Yes	Yes	Yes	Yes
Flag Register	Yes	Yes	Yes	Yes	IOPL	IOPL*
Control Registers	Yes	Yes	PL = 0	PL = 0	No	Yes
GDTR	Yes	Yes	PL = 0	Yes	No	Yes
IDTR	Yes	Yes	PL = 0	Yes	No	Yes
LDTR	No	No	PL = 0	Yes	No	No
TR	No	No	PL = 0	Yes	No	No
FPU Data Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Control Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Status Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Instruction Pointer	Yes	Yes	Yes	Yes	Yes	Yes
FPU Data Pointer	Yes	Yes	Yes	Yes	Yes	Yes
Debug Registers	Yes	Yes	PL = 0	PL = 0	No	No
Test Registers	Yes	Yes	PL = 0	PL = 0	No	No

NOTES:
PL = 0: The registers can be accessed only when the current privilege level is zero.
*IOPL: The PUSHF and POPF instructions are made I/O Privilege Level sensitive in Virtual 86 Mode.

2.1.6 COMPATIBILITY

VERY IMPORTANT NOTE:
COMPATIBILITY WITH FUTURE PROCESSORS

In the preceding register descriptions, note certain Intel486 Microprocessor register bits are Intel reserved. When reserved bits are called out, treat them as fully undefined. This is essential for your software compatibility with future processors! Follow the guidelines below:

1) **Do not depend on the states of any undefined bits when testing the values of defined register bits. Mask them out when testing.**
2) **Do not depend on the states of any undefined bits when storing them to memory or another register.**
3) **Do not depend on the ability to retain information written into any undefined bits.**
4) **When loading registers always load the undefined bits as zeros.**
5) **However, registers which have been previously stored may be reloaded without masking.**

Depending upon the values of undefined register bits will make your software dependent upon the unspecified Intel486 Microprocessor handling of these bits. Depending on undefined values risks making your software incompatible with future processors that define usages for the Intel486 Microprocessor-undefined bits. AVOID ANY SOFTWARE DEPENDENCE UPON THE STATE OF UNDEFINED Intel486 MICROPROCESSOR REGISTER BITS.

2.2 Instruction Set

The Intel486 Microprocessor instruction set can be divided into 11 categories of operations:

Data Transfer
Arithmetic
Shift/Rotate
String Manipulation
Bit Manipulation
Control Transfer
High Level Language Support
Operating System Support
Processor Control
Floating Point
Floating Point Control

The Intel486 Microprocessor instructions are listed in Section 10. Note that all floating point unit instruction mnemonics begin with an F.

All Intel486 Microprocessor instructions operate on either 0, 1, 2 or 3 operands; where an operand resides in a register, in the instruction itself or in memory. Most zero operand instructions (e.g., CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2 bytes long. Since the Intel486 Microprocessor has a 32-byte instruction queue, an average of 10 instructions will be prefetched. The use of two operands permits the following types of common instructions:

Register to Register
Memory to Register
Memory to Memory
Immediate to Register
Register to Memory
Immediate to Memory

The operands can be either 8, 16, or 32 bits long. As a general rule, when executing code written for the Intel486 or 386 Microprocessors (32-bit code), operands are 8 or 32 bits; when executing existing 80286 or 8086 code (16-bit code), operands are 8 or 16 bits. Prefixes can be added to all instructions which override the default length of the operands (i.e., use 32-bit operands for 16-bit code, or 16-bit operands for 32-bit code).

2.3 Memory Organization

Introduction

Memory on the Intel486 Microprocessor is divided up into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address, the high order byte at the high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address, the high-order byte at the highest address. The address of a word or dword is the byte address of the low-order byte.

In addition to these basic data types, the Intel486 Microprocessor supports two larger units of memory: pages and segments. Memory can be divided up into one or more variable length segments, which can be swapped to disk or shared between programs. Memory can also be organized into one or more 4 Kbyte pages. Finally, both segmentation and paging can be combined, gaining the advantages of both systems. The Intel486 Microprocessor supports both pages and segments in order to provide maximum flexibility to the system designer. Segmentation and paging are complementary. Segmentation is useful for organizing memory in logical modules, and as such is a tool for the application programmer, while pages are useful for the system programmer for managing the physical memory of a system.

2.3.1 ADDRESS SPACES

The Intel486 Microprocessor has three distinct address spaces: **logical, linear,** and **physical**. A **logical** address (also known as a **virtual** address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, DISPLACEMENT) discussed in Section 2.5.3 **Memory Addressing Modes** into an effective address. Since each task on the Intel486 Microprocessor has a maximum of 16K ($2^{14} - 1$) selectors, and offsets can be 4 gigabytes, (2^{32} bits) this gives a total of 2^{46} bits or 64 terabytes of **logical** address space per task. The programmer sees this virtual address space.

The segmentation unit translates the **logical** address space into a 32-bit **linear** address space. If the paging unit is not enabled then the 32-bit **linear** address corresponds to the **physical** address. The paging unit translates the **linear** address space into the **physical** address space. The **physical address** is what appears on the address pins.

The primary difference between Real Mode and Protected Mode is how the segmentation unit performs the translation of the **logical** address into the **linear** address. In Real Mode, the segmentation unit shifts the selector left four bits and adds the result to the offset to form the **linear** address. While in Protected Mode every selector has a **linear** base address associated with it. The **linear base** address is stored in one of two operating system tables (i.e., the Local Descriptor Table or Global Descriptor Table). The selector's **linear base** address is added to the offset to form the final **linear** address.

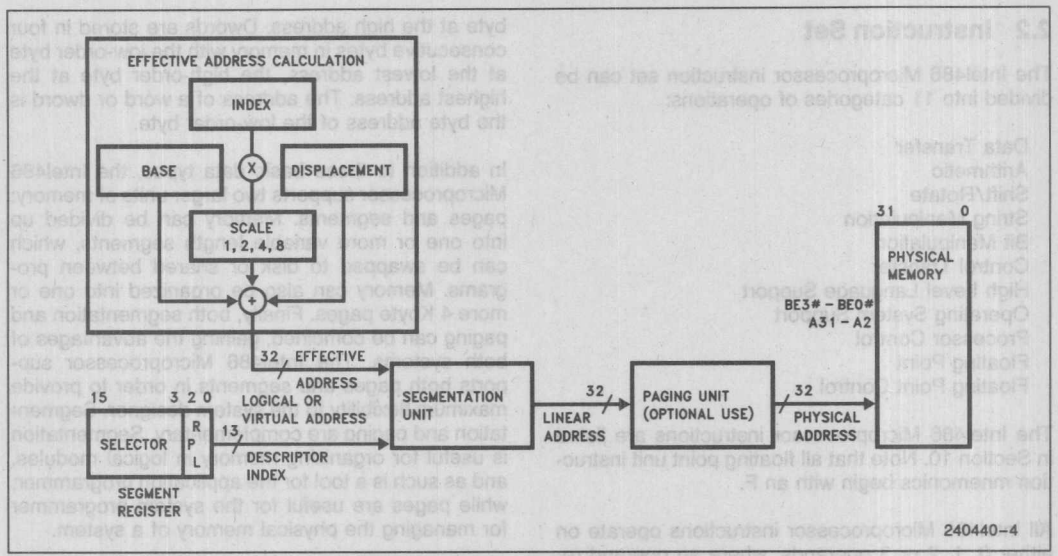


Figure 2.16. Address Translation

Figure 2.16 shows the relationship between the various address spaces.

2.3.2 SEGMENT REGISTER USAGE

The main data structure used to organize memory is the segment. On the Intel486 Microprocessor, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments: code and data, the segments are of variable size and can be as small as 1 byte or as large as 4 gigabytes (2^{32} bytes).

In order to provide compact instruction encoding, and increase processor performance, instructions do not need to explicitly specify which segment register is used. A default segment register is automatically chosen according to the rules of Table 2.11 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register; Stack references use the SS register and Instruction fetches use the CS register. The contents of the Instruction Pointer provide the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 2.11. The override prefixes also allow the use of the ES, FS and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero and create a system with a four gigabyte linear address space. This creates a system where the virtual address space is the same as the linear address space. Further details of segmentation are discussed in Section 4.1.

2.4 I/O Space

The Intel486 Microprocessor has two distinct physical address spaces: Memory and I/O. Generally, peripherals are placed in I/O space although the Intel486 Microprocessor also supports memory-mapped peripherals. The I/O space consists of 64 Kbytes, it can be divided into 64K 8-bit ports, 32K 16-bit ports, or 16K 32-bit ports, or any combination of ports which add up to less than 64 Kbytes. The 64K I/O address space refers to physical memory rather than linear address since I/O instructions do not go through the segmentation or paging hardware. The M/IO# pin acts as an additional address line thus allowing the system designer to easily determine which address space the processor is accessing.

The I/O ports are accessed via the IN and OUT I/O instructions, with the port address supplied as an immediate 8-bit constant in the instruction or in the DX register. All 8- and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/IO# pin to be driven low.

I/O port addresses 00F8H through 00FFH are reserved for use by Intel.

Table 2.11. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment Use	Segment Override Prefixes Possible
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA Instructions	SS	None
Source of POP, POPA, POPF, IRET, RET instructions	SS	None
Destination of STOS, MOVS, REP STOS, REP MOVS Instructions (DI is Base Register)	ES	None
Other Data References, with Effective Address Using Base Register of:		
[EAX]	DS	All
[EBX]	DS	
[ECX]	DS	
[EDX]	DS	
[ESI]	DS	
[EDI]	DS	
[EBP]	SS	
[ESP]	SS	

2.5 Addressing Modes

2.5.1 ADDRESSING MODES OVERVIEW

The Intel486 Microprocessor provides a total of 11 addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

2.5.2 REGISTER AND IMMEDIATE MODES

Two of the addressing modes provide for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8-, 16- or 32-bit general registers.

Immediate Operand Mode: The operand is included in the instruction as part of the opcode.

2.5.3 32-BIT MEMORY ADDRESSING MODES

The remaining 9 modes provide a mechanism for specifying the effective address of an operand. The linear address consists of two components: the segment base address and an effective address. The effective address is calculated by using combinations of the following four address elements:

DISPLACEMENT: An 8-, or 32-bit immediate value, following the instruction.

BASE: The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area.

INDEX: The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array, or a string of characters.

SCALE: The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. Scaled index

mode is especially useful for accessing arrays or structures.

Combinations of these 4 components make up the 9 additional addressing modes. There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions. The one exception is the simultaneous use of Base and Index components which requires one additional clock.

As shown in Figure 2.17, the effective address (EA) of an operand is calculated according to the following formula.

EA = Base Reg + (Index Reg * Scaling) + Displacement

Direct Mode: The operand's offset is contained as part of the instruction as an 8-, 16- or 32-bit displacement.
EXAMPLE: INC Word PTR [500]

Register Indirect Mode: A BASE register contains the address of the operand.
EXAMPLE: MOV [ECX], EDX

Based Mode: A BASE register's contents is added to a DISPLACEMENT to form the operand's offset.
EXAMPLE: MOV ECX, [EAX + 24]

Index Mode: An INDEX register's contents is added to a DISPLACEMENT to form the operand's offset.
EXAMPLE: ADD EAX, TABLE[ESI]

Scaled Index Mode: An INDEX register's contents is multiplied by a scaling factor which is added to a DISPLACEMENT to form the operand's offset.
EXAMPLE: IMUL EBX, TABLE[ESI*4],7

Based Index Mode: The contents of a BASE register is added to the contents of an INDEX register to form the effective address of an operand.
EXAMPLE: MOV EAX, [ESI] [EBX]

Based Scaled Index Mode: The contents of an INDEX register is multiplied by a SCALING factor and the result is added to the contents of a BASE register to obtain the operand's offset.
EXAMPLE: MOV ECX, [EDX*8] [EAX]

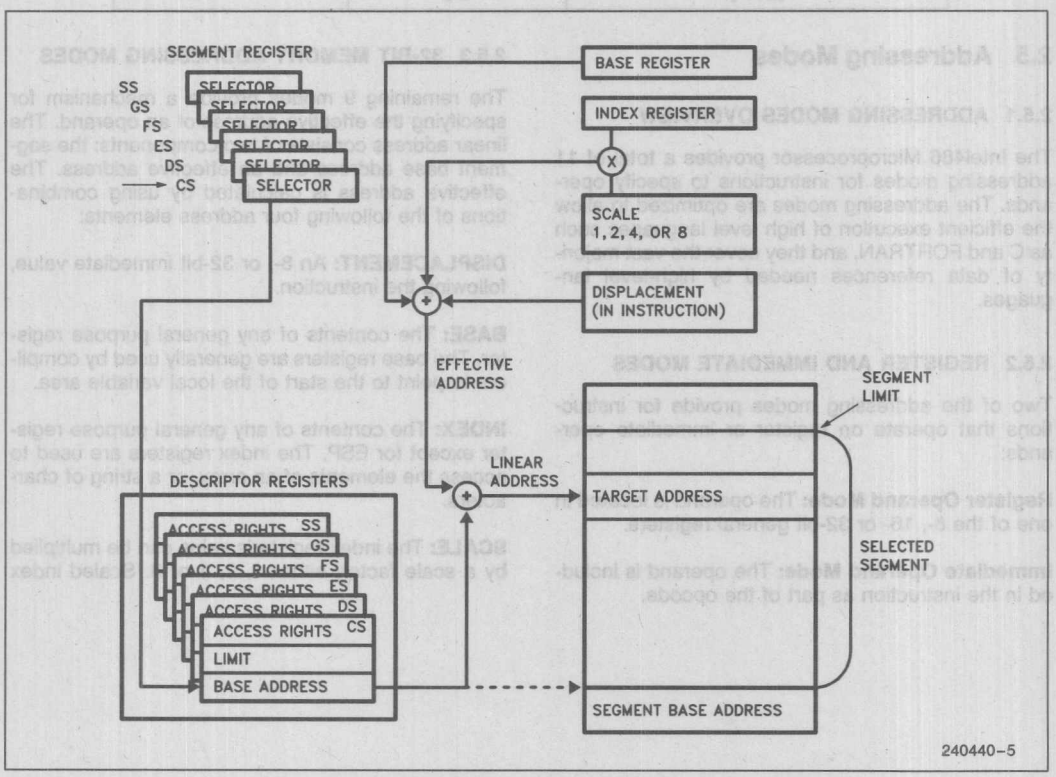


Figure 2.17. Addressing Mode Calculations

Based Index Mode with Displacement: The contents of an INDEX Register and a BASE register's contents and a DISPLACEMENT are all summed together to form the operand offset.
EXAMPLE: ADD EDX, [ESI] [EBP+00FFFFF0H]

Based Scaled Index Mode with Displacement: The contents of an INDEX register are multiplied by a SCALING factor, the result is added to the contents of a BASE register and a DISPLACEMENT to form the operand's offset.
EXAMPLE: MOV EAX, LOCALTABLE[EDI*4] [EBP+80]

2.5.4 DIFFERENCES BETWEEN 16- AND 32-BIT ADDRESSES

In order to provide software compatibility with the 80286 and the 8086, the Intel486 Microprocessor can execute 16-bit instructions in Real and Protected Modes. The processor determines the size of the instructions it is executing by examining the D bit in the CS segment Descriptor. If the D bit is 0 then all operand lengths and effective addresses are assumed to be 16 bits long. If the D bit is 1 then the default length for operands and addresses is 32 bits. In Real Mode the default size for operands and addresses is 16-bits.

Regardless of the default precision of the operands or addresses, the Intel486 Microprocessor is able to execute either 16- or 32-bit instructions. This is specified via the use of override prefixes. Two prefixes, the **Operand Size Prefix** and the **Address Length Prefix**, override the value of the D bit on an individual instruction basis. These prefixes are automatically added by Intel assemblers.

Example: The processor is executing in Real Mode and the programmer needs to access the EAX registers. The assembler code for this might be MOV EAX, 32-bit MEMORYOP, ASM486 Macro Assembler automatically determines that an Operand Size Prefix is needed and generates it.

Example: The D bit is 0, and the programmer wishes to use Scaled Index addressing mode to access an array. The Address Length Prefix allows the use of MOV DX, TABLE[ESI*2]. The assembler uses an Address Length Prefix since, with D=0, the default addressing mode is 16-bits.

Example: The D bit is 1, and the program wants to store a 16-bit quantity. The Operand Length Prefix is used to specify only a 16-bit value; MOV MEM16, DX.

The OPERAND LENGTH and Address Length Prefixes can be applied separately or in combination to any instruction. The Address Length Prefix does not allow addresses over 64 Kbytes to be accessed in Real Mode. A memory address which exceeds FFFFH will result in a General Protection Fault. An Address Length Prefix only allows the use of the additional Intel486 Microprocessor addressing modes.

When executing 32-bit code, the Intel486 Microprocessor uses either 8-, or 32-bit displacements, and any register can be used as base or index registers. When executing 16-bit code, the displacements are either 8, or 16 bits, and the base and index register conform to the 80286 model. Table 2.12 illustrates the differences.

2.6 Data Formats

2.6.1 DATA TYPES

The Intel486 Microprocessor can support a wide variety of data types. In the following descriptions, the on-chip floating point unit (FPU) consists of the floating point registers. The central processing unit (CPU) consists of the base architecture registers.

2.6.1.1 Unsigned Data Types

The FPU does not support unsigned data types. Refer to Table 2.13.

Byte: Unsigned 8-bit quantity

Word: Unsigned 16-bit quantity

Dword: Unsigned 32-bit quantity

The least significant bit (LSB) in a byte is bit 0, and the most significant bit is 7.

Table 2.12. BASE and INDEX Registers for 16- and 32-Bit Addresses

	16-Bit Addressing	32-Bit Addressing
BASE REGISTER	BX,BP	Any 32-bit GP Register
INDEX REGISTER	SI,DI	Any 32-bit GP Register Except ESP
SCALE FACTOR	none	1, 2, 4, 8
DISPLACEMENT	0, 8, 16 bits	0, 8, 32 bits

2.6.1.2 Signed Data Types

All signed data types assume 2's complement notation. The signed data types contain two fields, a sign bit and a magnitude. The sign bit is the most significant bit (MSB). The number is negative if the sign bit is 1. If the sign bit is 0, the number is positive. The magnitude field consists of the remaining bits in the number. Refer to Table 2.13.

8-bit Integer: Signed 8-bit quantity

16-bit Integer: Signed 16-bit quantity

32-bit Integer: Signed 32-bit quantity

64-bit Integer: Signed 64-bit quantity

The FPU only supports 16-, 32- and 64-bit integers. The CPU only supports 8-, 16- and 32-bit integers.

2.6.1.3 Floating Point Data Types

Floating point data type in the Intel486 Microprocessor contain three fields, sign, significand and exponent. The sign field is one bit and is the MSB of the floating point number. The number is negative if the sign bit is 1. If the sign bit is 0, the number is positive. The significand gives the significant bits of the number. The exponent field contains the power of 2 needed to scale the significand. Refer to Table 2.13.

Only the FPU supports floating point data types.

Single Precision Real: 23-bit significand and 8-bit exponent. 32 bits total.

Double Precision Real: 52-bit significand and 11-bit exponent. 64 bits total.

Extended Precision Real: 64-bit significand and 15-bit exponent. 80 bits total.

2.6.1.4 BCD Data Types

The Intel486 Microprocessor supports packed and unpacked binary coded decimal (BCD) data types. A packed BCD data type contains two digits per byte, the lower digit is in bits 0–3 and the upper digit in bits 4–7. An unpacked BCD data type contains 1 digit per byte stored in bits 0–3.

The CPU supports 8-bit packed and unpacked BCD data types. The FPU only supports 80-bit packed BCD data types. Refer to Table 2.13.

2.6.1.5 String Data Types

A string data type is a contiguous sequence of bits, bytes, words or dwords. A string may contain between 1 byte and 4 Gbytes. Refer to Table 2.14.

String data types are only supported by the CPU.

Byte String: Contiguous sequence of bytes.

Word String: Contiguous sequence of words.

Dword String: Contiguous sequence of dwords.

Bit String: A set of contiguous bits. In the Intel486 Microprocessor bit strings can be up to 4 gigabits long.

2.6.1.6 ASCII Data Types

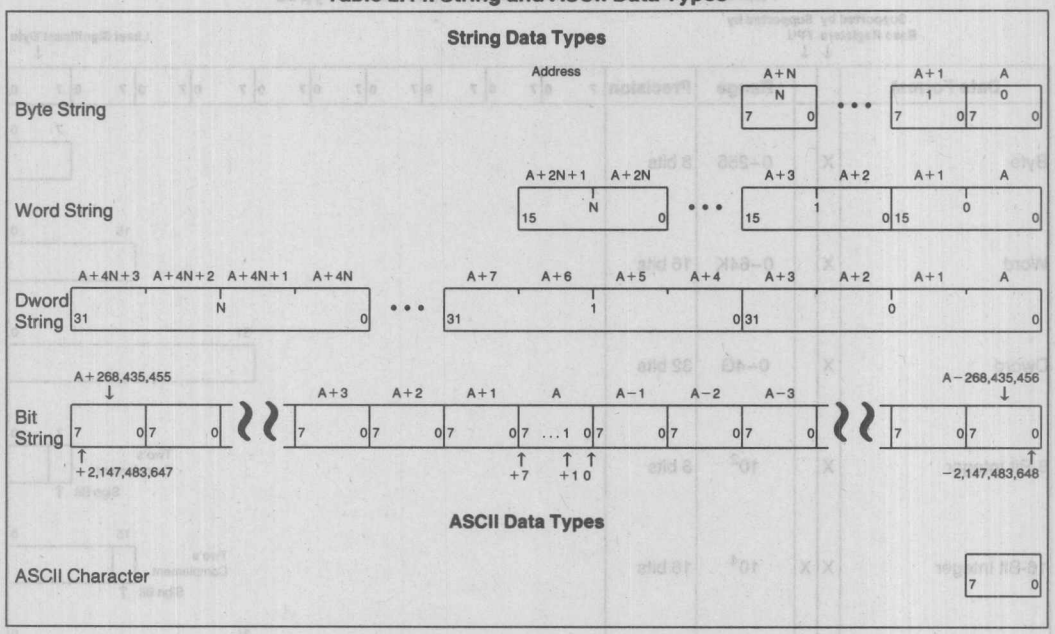
The Intel486 Microprocessor supports ASCII (American Standard Code for Information Interchange) strings and can perform arithmetic operations (such as addition and division) on ASCII data. Refer to Table 2.14.

Table 2.13. Intel486™ Microprocessor Data Types

Data Format	Supported by Base Registers ↓	Supported by FPU ↓	Range	Precision	Least Significant Byte ↓
Byte	X		0–255	8 bits	7 0
Word	X		0–64K	16 bits	15 0
Dword	X		0–4G	32 bits	31 0
8-Bit Integer	X		10^2	8 bits	7 0 Two's Complement Sign Bit ↑
16-Bit Integer	X	X	10^4	16 bits	15 0 Two's Complement Sign Bit ↑
32-Bit Integer	X	X	10^9	32 bits	31 0 Two's Complement Sign Bit ↑
64-Bit Integer		X	10^{19}	64 bits	63 0 Two's Complement Sign Bit ↑
8-Bit Unpacked BCD	X		0–9	1 Digit	7 0 One BCD Digit per Byte
8-Bit Packed BCD	X		0–9	2 Digits	7 0 Two BCD Digits per Byte
80-Bit Packed BCD		X	$\pm 10^{\pm 18}$	18 Digits	79 72 0 Ignored ↑ Sign Bit
Single Precision Real		X	$\pm 10^{\pm 38}$	24 Bits	31 23 0 Sign Bit ↑ Biased Exp. Significand
Double Precision Real		X	$\pm 10^{\pm 308}$	53 Bits	63 52 0 Sign Bit ↑ Biased Exp. Significand
Extended Precision Real		X	$\pm 10^{\pm 4932}$	64 Bits	79 63 0 ↑ Sign Bit Biased Exp. 1 Significand

2

Table 2.14. String and ASCII Data Types



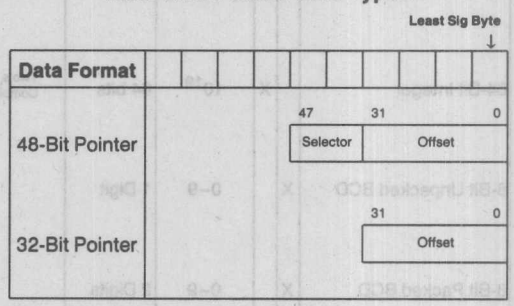
2.6.1.7 Pointer Data Types

A pointer data type contains a value that gives the address of a piece of data. The Intel486 Microprocessor supports two types of pointers. Refer to Table 2.15.

48-bit Pointer: 16-bit selector and 32-bit offset

32-bit Pointer: 32-bit offset

Table 2.15. Pointer Data Types



2.6.2 LITTLE ENDIAN vs BIG ENDIAN DATA FORMATS

The Intel486 Microprocessor, as well as all other members of the 86 architecture use the "little-endian" method for storing data types that are larger than one byte. Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address and the high order byte at the high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address and the high order byte at the highest address. The address of a word or dword data item is the byte address of the low-order byte.

Figure 2.18 illustrates the differences between the big-endian and little-endian formats for dwords. The 32 bits of data are shown with the low order bit numbered bit 0 and the high order bit numbered 32. Big-endian data is stored with the high-order bits at the lowest addressed byte. Little-endian data is stored with the high-order bits in the highest addressed byte.

The Intel486 Microprocessor has two instructions which can convert 16- or 32-bit data between the two byte orderings. BSWAP (byte swap) handles four byte values and XCHG (exchange) handles two byte values.

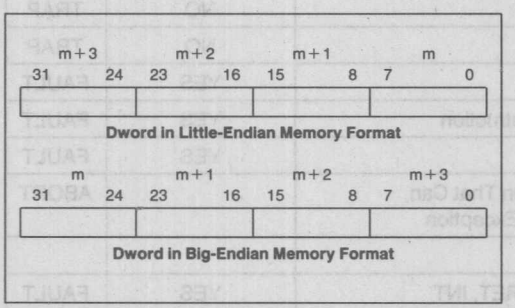


Figure 2.18. Big vs Little Endian Memory Format

2.7 Interrupts

2.7.1 INTERRUPTS AND EXCEPTIONS

Interrupts and exceptions alter the normal program flow, in order to handle external events, to report errors or exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT N instruction, the processor treats software interrupts as exceptions.

Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately **after** the interrupted instruction. Sections 2.7.3 and 2.7.4 discuss the differences between Maskable and Non-Maskable interrupts.

Exceptions are classified as faults, traps, or aborts depending on the way they are reported, and whether or not restart of the instruction causing the exception is supported. **Faults** are exceptions that are detected and serviced **before** the execution of the faulting instruction. A fault would occur in a virtual memory system, when the processor referenced a page or a segment which was not present. The operating system would fetch the page or segment from disk, and then the Intel486 Microprocessor would restart the instruction. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. User defined interrupts are examples of traps. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined. Aborts are used to report severe errors, such as a hardware error, or illegal values in system tables.

Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand, the return address from an exception fault routine will always point at the instruction causing the exception and include any leading instruction prefixes. Table 2.16 summarizes the possible interrupts for the Intel486 Microprocessor and shows where the return address points.

The Intel486 Microprocessor has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. In Real Mode (see Section 3.1), the vectors are 4 byte quantities, a Code Segment plus a 16-bit offset; in Protected Mode, the interrupt vectors are 8 byte quantities, which are put in an Interrupt Descriptor Table (see Section 4.3.3.4). Of the 256 possible interrupts, 32 are reserved for use by Intel, the remaining 224 are free to be used by the system designer.

2.7.2 INTERRUPT PROCESSING

When an interrupt occurs the following actions happen. First, the current program address and the Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is sup-

plied to the Intel486 Microprocessor which identifies the appropriate entry in the interrupt table. The table contains the starting address of the interrupt service routine. Then, the user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the Intel486 Microprocessor in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

2.7.3 MASKABLE INTERRUPT

Maskable interrupts are the most common way used by the Intel486 Microprocessor to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled high and the Interrupt Flag bit (IF) is enabled. The processor only responds to interrupts between instructions, (REPeat String instructions, have an "interrupt window", between memory moves, which allows interrupts during long string moves). When an interrupt occurs the processor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt, (one of 224 user defined interrupts). The exact nature of the interrupt sequence is discussed in Section 7.2.10.

Table 2.16. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	YES	FAULT
Debug Exception	1	Any Instruction	YES	TRAP*
NMI Interrupt	2	INT 2 or NMI	NO	NMI
One Byte Interrupt	3	INT	NO	TRAP
Interrupt on Overflow	4	INTO	NO	TRAP
Array Bounds Check	5	BOUND	YES	FAULT
Invalid OP-Code	6	Any Illegal Instruction	YES	FAULT
Device Not Available	7	ESC, WAIT	YES	FAULT
Double Fault	8	Any Instruction That Can Generate an Exception		ABORT
Intel Reserved	9			
Invalid TSS	10	JMP, CALL, IRET, INT	YES	FAULT
Segment Not Present	11	Segment Register Instructions	YES	FAULT
Stack Fault	12	Stack References	YES	FAULT
General Protection Fault	13	Any Memory Reference	YES	FAULT
Page Fault	14	Any Memory Access or Code Fetch	YES	FAULT
Intel Reserved	15			
Floating Point Error	16	Floating Point, WAIT	YES	FAULT
Alignment Check Interrupt	17	Unaligned Memory Access	YES	FAULT
Intel Reserved	18–31			
Two Byte Interrupt	0–255	INT n	NO	TRAP

*Some debug exceptions may report both traps on the previous instruction, and faults on the next instruction.

The IF bit in the EFLAG registers is reset when an interrupt is being serviced. This effectively disables servicing additional interrupts during an interrupt service routine. However, the IF may be set explicitly by the interrupt handler, to allow the nesting of interrupts. When an IRET instruction is executed the original state of the IF is restored.

2.7.4 NON-MASKABLE INTERRUPT

Non-maskable interrupts provide a method of servicing very high priority interrupts. A common example of the use of a non-maskable interrupt (NMI) would be to activate a power failure routine. When the NMI input is pulled high it causes an interrupt with an internally supplied vector value of 2. Unlike a normal hardware interrupt, no interrupt acknowledgment sequence is performed for an NMI.

While executing the NMI servicing procedure, the Intel486 Microprocessor will not service further NMI requests until an interrupt return (IRET) instruction is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The IF bit is cleared at the beginning of an NMI interrupt to inhibit further INTR interrupts.

2.7.5 SOFTWARE INTERRUPTS

A third type of interrupt/exception for the Intel486 Microprocessor is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table.

A special case of the two byte software interrupt INT n is the one byte INT 3, or breakpoint interrupt. By inserting this one byte instruction in a program, the user can set breakpoints in his program as a debugging tool.

A final type of software interrupt is the single step interrupt. It is discussed in Section 9.2.

2.7.6 INTERRUPT AND EXCEPTION PRIORITIES

Interrupts are externally-generated events. Maskable Interrupts (on the INTR input) and Non-Maskable Interrupts (on the NMI input) are recognized at instruction boundaries. When NMI and maskable INTR are **both** recognized at the **same** instruction boundary, the Intel486 Microprocessor invokes the NMI service routine first. If, after the NMI service routine has been invoked, maskable interrupts are still enabled, then the Intel486 Microprocessor will invoke the appropriate interrupt service routine.

Table 2.17a. Intel486™ Microprocessor Priority for Invoking Service Routines in Case of Simultaneous External Interrupts

1. NMI
2. INTR

Exceptions are internally-generated events. Exceptions are detected by the Intel486 Microprocessor if, in the course of executing an instruction, the Intel486 Microprocessor detects a problematic condition. The Intel486 Microprocessor then immediately invokes the appropriate exception service routine. The state of the Intel486 Microprocessor is such that the instruction causing the exception can be restarted. If the exception service routine has taken care of the problematic condition, the instruction will execute without causing the same exception.

It is possible for a single instruction to generate several exceptions (for example, transferring a single operand could generate two page faults if the operand location spans two "not present" pages). However, only one exception is generated upon each attempt to execute the instruction. Each exception service routine should correct its corresponding exception, and restart the instruction. In this manner, exceptions are serviced until the instruction executes successfully.

As the Intel486 Microprocessor executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2.17b. This cycle is repeated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

Table 2.17b. Sequence of Exception Checking

Consider the case of the Intel486 Microprocessor having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed:

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
3. Check for external NMI and INTR.
4. Check for Segmentation Faults that prevented fetching the entire next instruction (exceptions 11 or 13).
5. Check for Page Faults that prevented fetching the entire next instruction (exception 14).
6. Check for Faults decoding the next instruction (exception 6 if illegal opcode; exception 6 if in Real Mode or in Virtual 8086 Mode and attempting to execute an instruction for Protected Mode only (see Section 4.6.4); or exception 13 if instruction is longer than 15 bytes, or privilege violation in Protected Mode (i.e., not at IOPL or at CPL=0).
7. If WAIT opcode, check if TS=1 and MP=1 (exception 7 if both are 1).
8. If opcode for Floating Point Unit, check if EM=1 or TS=1 (exception 7 if either are 1).
9. If opcode for Floating Point Unit (FPU), check FPU error status (exception 16 if error status is asserted).
10. Check in the following order for each memory reference required by the instruction:
    a. Check for Segmentation Faults that prevent transferring the entire memory quantity (exceptions 11, 12, 13).
    b. Check for Page Faults that prevent transferring the entire memory quantity (exception 14).

NOTE:
The order stated supports the concept of the paging mechanism being "underneath" the segmentation mechanism. Therefore, for any given code or data reference in memory, segmentation exceptions are generated before paging exceptions are generated.

2.7.7 INSTRUCTION RESTART

The Intel486 Microprocessor fully supports restarting all instructions after faults. If an exception is detected in the instruction to be executed (exception categories 4 through 10 in Table 2.17b), the Intel486 Microprocessor invokes the appropriate exception service routine. The Intel486 Microprocessor is in a state that permits restart of the instruction, for all cases but those in Table 2.17c. Note that all such cases are easily avoided by proper design of the operating system.

Table 2.17c. Conditions Preventing Instruction Restart

An instruction causes a task switch to a task whose Task State Segment is **partially** "not present". (An entirely "not present" TSS is restartable.) Partially present TSS's can be avoided either by keeping the TSS's of such tasks present in memory, or by aligning TSS segments to reside entirely within a single 4K page (for TSS segments of 4 Kbytes or less).

NOTE:
These conditions are avoided by using the operating system designs mentioned in this table.

2.7.8 DOUBLE FAULT

A Double Fault (exception 8) results when the processor attempts to invoke an exception service routine for the segment exceptions (10, 11, 12 or 13), but in the process of doing so, detects an exception other than a Page Fault (exception 14).

A Double Fault (exception 8) will also be generated when the processor attempts to invoke the Page Fault (exception 14) service routine, and detects an exception other than a second Page Fault. In any functional system, the entire Page Fault service routine must remain "present" in memory.

When a Double Fault occurs, the Intel486 Microprocessor invokes the exception service routine for exception 8.

2.7.9 FLOATING POINT INTERRUPT VECTORS

Several interrupt vectors of the Intel486 Microprocessor are used to report exceptional conditions while executing numeric programs in either real or protected mode. Table 2.18 shows these interrupts and their causes.

PRELIMINARY

Table 2.18. Interrupt Vectors Used by FPU

Interrupt Number	Cause of Interrupt
7	A Floating Point instruction was encountered when EM or TS of the Intel486™ Processor control register zero (CR0) was set. EM = 1 indicates that software emulation of the instruction is required. When TS is set, either a Floating Point or WAIT instruction causes interrupt 7. This indicates that the current FPU context may not belong to the current task.
13	The first word or doubleword of a numeric operand is not entirely within the limit of its segment. The return address pushed onto the stack of the exception handler points at the Floating Point instruction that caused the exception, including any prefixes. The FPU has not executed this instruction; the instruction pointer and data pointer register refer to a previous, correctly executed instruction.
16	The previous numerics instruction caused an unmasked exception. The address of the faulty instruction and the address of its operand are stored in the instruction pointer and data pointer registers. Only Floating Point and WAIT instructions can cause this interrupt. The Intel486™ Processor return address pushed onto the stack of the exception handler points to a WAIT or Floating Point instruction (including prefixes). This instruction can be restarted after clearing the exception condition in the FPU. The FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE instructions cannot cause this interrupt.

3.0 REAL MODE ARCHITECTURE

3.1 Real Mode Introduction

When the processor is reset or powered up it is initialized in Real Mode. Real Mode has the same base architecture as the 8086, but allows access to the 32-bit register set of the Intel486 Microprocessor. The addressing mechanism, memory size, interrupt handling, are all identical to the Real Mode on the 80286.

All of the Intel486 Microprocessor instructions are available in Real Mode (except those instructions listed in Section 4.6.4). The default operand size in Real Mode is 16 bits, just like the 8086. In order to use the 32-bit registers and addressing modes, override prefixes must be used. In addition, the segment size on the Intel486 Microprocessor in Real Mode is 64 Kbytes so 32-bit effective addresses must have a value less the 0000FFFFH. The primary purpose of Real Mode is to set up the processor for Protected Mode Operation.

The LOCK prefix on the Intel486 Microprocessor, even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the Intel486 Microprocessor in Protected Mode and Virtual 8086 Mode. Paging makes it impossible to guarantee that repeated string instructions can be LOCKed. The Intel486 Microprocessor can't require that all pages holding the string be physically present in memory. Hence, a Page Fault (exception 14) might have to be taken during the repeated string instruction. Therefore the LOCK prefix can't be supported during repeated string instructions.

These are the only instruction forms where the LOCK prefix is legal on the Intel486 Microprocessor:

Opcode	Operands (Dest, Source)
BIT Test and SET/RESET/COMPLEMENT	Mem, Reg/immed
XCHG	Reg, Mem
XCHG	Mem, Reg
ADD, OR, ADC, SBB, AND, SUB, XOR	Mem, Reg/immed
NOT, NEG, INC, DEC	Mem
CMPXCHG, XADD	Mem, Reg

An exception 6 will be generated if a LOCK prefix is placed before any instruction form or opcode not listed above. The LOCK prefix allows indivisible read/modify/write operations on memory operands using the instructions above. For example, even the ADD Reg, Mem is not LOCKable, because the Mem operand is not the destination (and therefore no memory read/modify/operation is being performed).

Since, on the Intel486 Microprocessor, repeated string instructions are not LOCKable, it is not possible to LOCK the bus for a long period of time. Therefore, the LOCK prefix is not IOPL-sensitive on the Intel486 Microprocessor. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed above.

3.2 Memory Addressing

In Real Mode the maximum memory size is limited to 1 megabyte. Thus, only address lines A2–A19 are active. (Exception, after RESET address lines A20–A31 are high during CS-relative memory cycles until an intersegment jump or call is executed (see Section 6.5)).

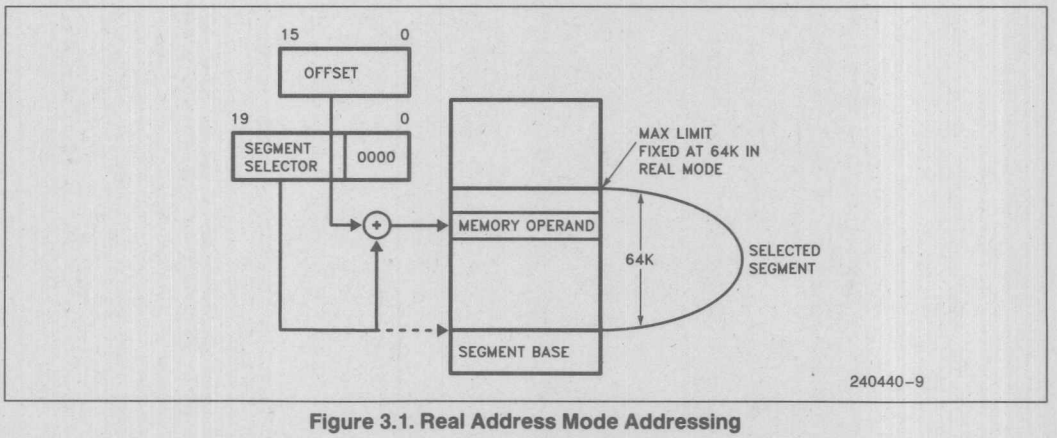


Figure 3.1. Real Address Mode Addressing

Since paging is not allowed in Real Mode the linear addresses are the same as physical addresses. Physical addresses are formed in Real Mode by adding the contents of the appropriate segment register which is shifted left by four bits to an effective address. This addition results in a physical address from 00000000H to 0010FFEFH. This is compatible with 80286 Real Mode. Since segment registers are shifted left by 4 bits, Real Mode segments always start on 16 byte boundaries.

All segments in Real Mode are exactly 64 Kbytes long, and may be read, written, or executed. The Intel486 Microprocessor will generate an exception 13 if a data operand or instruction fetch occurs past the end of a segment (i.e., if an operand has an offset greater than FFFFH, for example a word with a low byte at FFFFH and the high byte at 0000H).

Segments may be overlapped in Real Mode. Thus, if a particular segment does not use all 64 Kbytes another segment can be overlayed on top of the unused portion of the previous segment. This allows the programmer to minimize the amount of physical memory needed for a program.

3.3 Reserved Locations

There are two fixed areas in memory which are reserved in Real address mode: system initialization area and the interrupt table area. Locations 00000H through 003FFH are reserved for interrupt vectors. Each one of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations FFFFFFF0H through FFFFFFFFH are reserved for system initialization.

3.4 Interrupts

Many of the exceptions shown in Table 2.16 and discussed in Section 2.7 are not applicable to Real Mode operation, in particular exceptions 10, 11, 14, 17, will not happen in Real Mode. Other exceptions have slightly different meanings in Real Mode; Table 3.1 identifies these exceptions.

3.5 Shutdown and Halt

The HLT instruction stops program execution and prevents the processor from using the local bus until restarted. Either NMI, INTR with interrupts enabled (IF=1), or RESET will force the Intel486 Microprocessor out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

As in the case in protected mode, the shutdown will occur when a severe error is detected that prevents further processing. In Real Mode, shutdown can occur under two conditions:

An interrupt or an exception occur (exceptions 8 or 13) and the interrupt vector is larger than the Interrupt Descriptor Table (i.e., there is not an interrupt handler for the interrupt).

A CALL, INT or PUSH instruction attempts to wrap around the stack segment when SP is not even (i.e., pushing a value on the stack when SP = 0001 resulting in a stack segment greater than FFFFH).

An NMI input can bring the processor out of shutdown if the Interrupt Descriptor Table limit is large enough to contain the NMI interrupt vector (at least 0017H) and the stack has enough room to contain the vector and flag information (i.e., SP is greater than 0005H). If these conditions are not met, the Intel486 CPU is unable to execute the NMI and executes another shutdown cycle. In this case, the processor remains in the shutdown and can only exit via the RESET input.

Table 3.1. Exceptions with Different Meanings in Real Mode (see Table 2.16)

Function	Interrupt Number	Related Instructions	Return Address Location
Interrupt table limit too small	8	INT Vector is not within table limit	Before Instruction
CS, DS, ES, FS, GS Segment overrun exception	13	Word memory reference beyond offset = FFFFH. An attempt to execute past the end of CS segment.	Before Instruction
SS Segment overrun exception	12	Stack Reference beyond offset = FFFFH	Before Instruction

4.0 PROTECTED MODE ARCHITECTURE

4.1 Introduction

The complete capabilities of the Intel486 Microprocessor are unlocked when the processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to four gigabytes (2^{32} bytes) and allows the running of virtual memory programs of almost unlimited size (64 terabytes or 2^{46} bytes). In addition Protected Mode allows the Intel486 Microprocessor to run all of the existing 8086, 80286 and 386 microprocessor software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions especially optimized for supporting multitasking operating systems. The base architecture of the Intel486 Microprocessor remains the same, the registers, instructions, and addressing modes described in the previous sections are retained. The main difference between Protected Mode, and Real Mode from a programmer's view is the increased address space, and a different addressing mechanism.

4.2 Addressing Mechanism

Like Real Mode, Protected Mode uses two components to form the logical address, a 16-bit selector is used to determine the linear base address of a segment, the base address is added to a 32-bit effective address to form a 32-bit linear address. The linear address is then either used as the 32-bit physical address, or if paging is enabled the paging mechanism maps the 32-bit linear address into a 32-bit physical address.

The difference between the two modes lies in calculating the base address. In Protected Mode the selector is used to specify an index into an operating system defined table (see Figure 4.1). The table contains the 32-bit base address of a given segment. The physical address is formed by adding the base address obtained from the table to the offset.

Paging provides an additional memory management mechanism which operates only in Protected Mode. Paging provides a means of managing the very large segments of the Intel486 Microprocessor. As such, paging operates beneath segmentation. The paging mechanism translates the protected linear address which comes from the segmentation unit into a physical address. Figure 4.2 shows the complete Intel486 Microprocessor addressing mechanism with paging enabled.

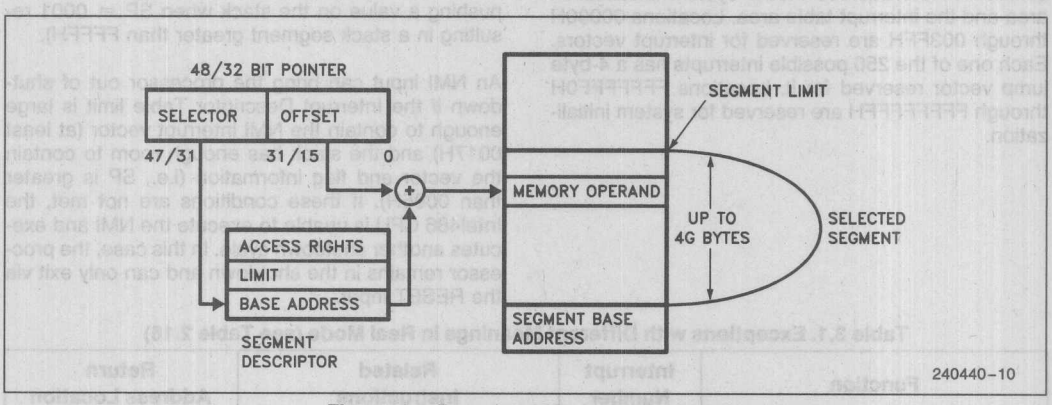


Figure 4.1. Protected Mode Addressing

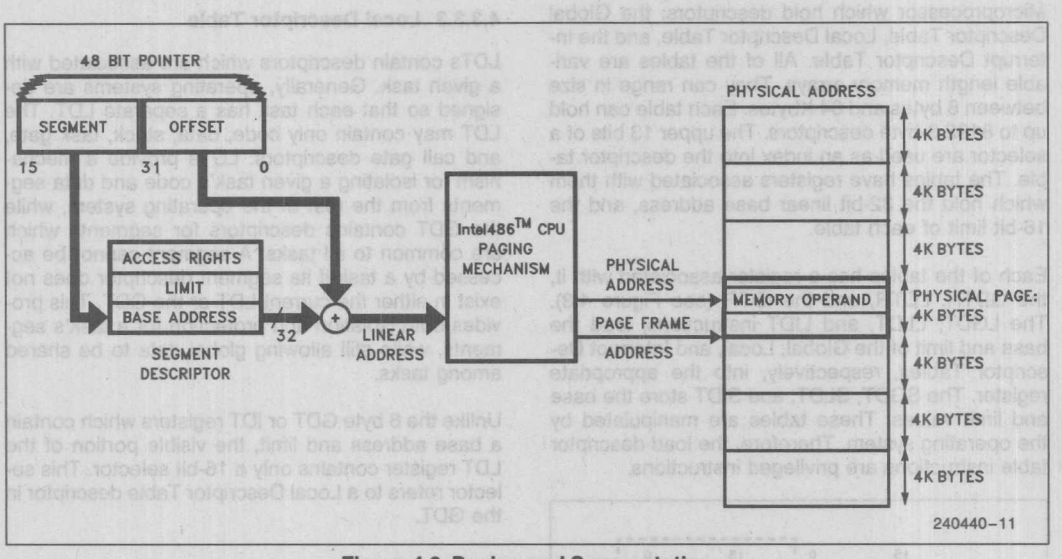


Figure 4.2. Paging and Segmentation

4.3 Segmentation

4.3.1 SEGMENTATION INTRODUCTION

Segmentation is one method of memory management. Segmentation provides the basis for protection. Segments are used to encapsulate regions of memory which have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about a segment is stored in an 8 byte data structure called a descriptor. All of the descriptors in a system are contained in tables recognized by hardware.

4.3.2 TERMINOLOGY

The following terms are used throughout the discussion of descriptors, privilege levels and protection:

PL: Privilege Level—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged. More privileged levels are numerically smaller than less privileged levels.

RPL: Requestor Privilege Level—The privilege level of the original supplier of the selector. RPL is determined by the **least two** significant bits of a selector.

DPL: Descriptor Privilege Level—This is the least privileged level at which a task may access that descriptor (and the segment associated with that descriptor). Descriptor Privilege Level is determined by bits 6:5 in the Access Right Byte of a descriptor.

CPL: Current Privilege Level—The privilege level at which a task is currently executing, which equals the privilege level of the code segment being executed. CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.

EPL: Effective Privilege Level—The effective privilege level is the least privileged of the RPL and DPL. Since smaller privilege level **values** indicate greater privilege, EPL is the numerical maximum of RPL and DPL.

Task: One instance of the execution of a program. Tasks are also referred to as processes.

4.3.3 DESCRIPTOR TABLES

4.3.3.1 Descriptor Tables Introduction

The descriptor tables define all of the segments which are used in an Intel486 Microprocessor system. There are three types of tables on the Intel486

Microprocessor which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays. They can range in size between 8 bytes and 64 Kbytes. Each table can hold up to 8192 8-byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address, and the 16-bit limit of each table.

Each of the tables has a register associated with it, the GDTR, LDTR, and the IDTR (see Figure 4.3). The LGDT, LLDT, and LIDT instructions, load the base and limit of the Global, Local, and Interrupt Descriptor Tables, respectively, into the appropriate register. The SGDT, SLDT, and SIDT store the base and limit values. These tables are manipulated by the operating system. Therefore, the load descriptor table instructions are privileged instructions.

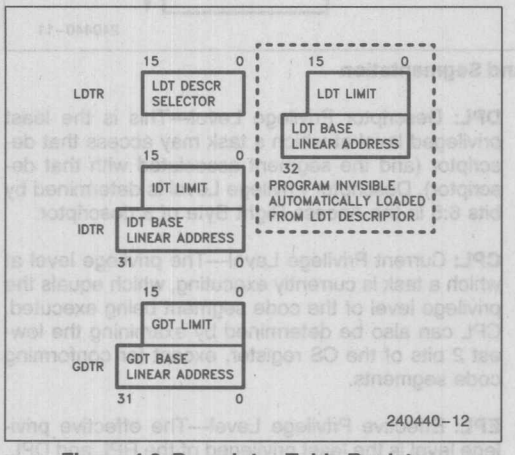


Figure 4.3. Descriptor Table Registers

4.3.3.2 Global Descriptor Table

The Global Descriptor Table (GDT) contains descriptors which are possibly available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for descriptors which are used for servicing interrupts (i.e., interrupt and trap descriptors). Every Intel486 Microprocessor system contains a GDT. Generally the GDT contains code and data segments used by the operating systems and task state segments, and descriptors for the LDTs in a system.

The first slot of the Global Descriptor Table corresponds to the null selector and is not used. The null selector defines a null pointer value.

4.3.3.3 Local Descriptor Table

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks. A segment cannot be accessed by a task if its segment descriptor does not exist in either the current LDT or the GDT. This provides both isolation and protection for a task's segments, while still allowing global data to be shared among tasks.

Unlike the 6 byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT.

4.3.3.4 Interrupt Descriptor Table

The third table needed for Intel486 Microprocessor systems is the Interrupt Descriptor Table. (See Figure 4.4.) The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced via INT instructions, external interrupt vectors, and exceptions. (See Section 2.7 **Interrupts).**

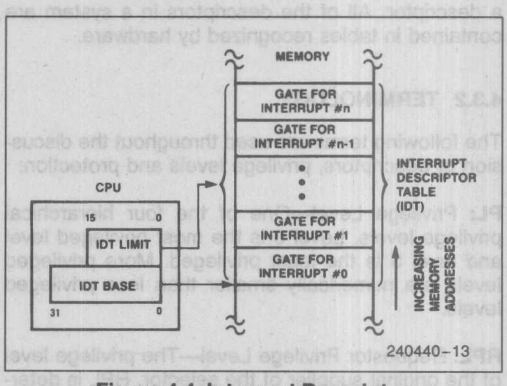


Figure 4.4. Interrupt Descriptor Table Register Use

4.3.4 DESCRIPTORS

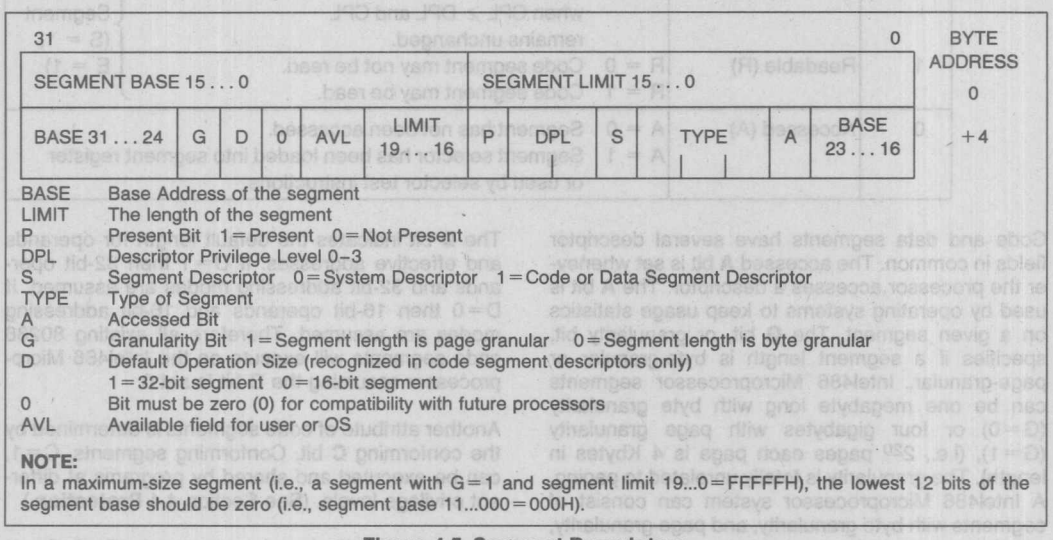
4.3.4.1 Descriptor Attribute Bits

The object to which the segment selector points to is called a descriptor. Descriptors are eight byte quantities which contain attributes about a given region of linear address space (i.e., a segment). These attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or 32-bit), and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 4.5 shows the general format of a descriptor. All segments on the Intel486 Microprocessor have three attribute fields in common: the **P** bit, the **DPL** bit, and the **S** bit. The Present **P** bit is 1 if the segment is loaded in physical memory, if P=0 then any attempt to access this segment causes a not present exception (exception 11). The Descriptor Privilege Level **DPL** is a two-bit field which specifies the protection level 0–3 associated with a segment.

The Intel486 Microprocessor has two main categories of segments: system segments and non-system segments (for code and data). The segment **S** bit in the segment descriptor determines if a given segment is a system segment or a code or data segment. If the S bit is 1 then the segment is either a code or data segment, if it is 0 then the segment is a system segment.

4.3.4.2 Intel486™ CPU Code, Data Descriptors (S=1)

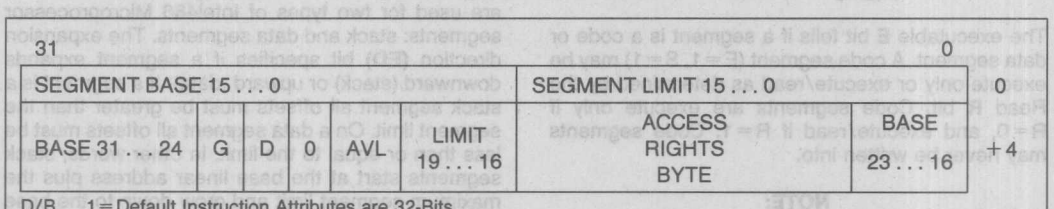
Figure 4.6 shows the general format of a code and data descriptor and Table 4.1 illustrates how the bits in the Access Rights Byte are interpreted.



BASE	Base Address of the segment
LIMIT	The length of the segment
P	Present Bit 1=Present 0=Not Present
DPL	Descriptor Privilege Level 0–3
S	Segment Descriptor 0=System Descriptor 1=Code or Data Segment Descriptor
TYPE	Type of Segment
A	Accessed Bit
G	Granularity Bit 1=Segment length is page granular 0=Segment length is byte granular
D	Default Operation Size (recognized in code segment descriptors only) 1=32-bit segment 0=16-bit segment
0	Bit must be zero (0) for compatibility with future processors
AVL	Available field for user or OS

NOTE:
In a maximum-size segment (i.e., a segment with G=1 and segment limit 19...0=FFFFFH), the lowest 12 bits of the segment base should be zero (i.e., segment base 11...000=000H).

Figure 4.5. Segment Descriptors



D/B	1=Default Instruction Attributes are 32-Bits 0=Default Instruction Attributes are 16-Bits
AVL	Available field for user or OS
G	Granularity Bit 1=Segment length is page granular 0=Segment length is byte granular
0	Bit must be zero (0) for compatibility with future processors

Figure 4.6. Segment Descriptors

Table 4.1. Access Rights Byte Definition for Code and Data Descriptions

	Bit Position	Name	Function		
	7	Present (P)	P = 1	Segment is mapped into physical memory.	
			P = 0	No mapping to physical memory exits, base and limit are not used.	
	6–5	Descriptor Privilege Level (DPL)		Segment privilege attribute used in privilege tests.	
	4	Segment Descriptor (S)	S = 1	Code or Data (includes stacks) segment descriptor.	
			S = 0	System Segment Descriptor or Gate Descriptor.	
Type Field Definition	3	Executable (E)	E = 0	Descriptor type is data segment:	If Data Segment (S = 1, E = 0)
	2	Expansion Direction (ED)	ED = 0	Expand up segment, offsets must be ≤ limit.	
			ED = 1	Expand down segment, offsets must be > limit.	
	1	Writeable (W)	W = 0	Data segment may not be written into.	
			W = 1	Data segment may be written into.	
	3	Executable (E)	E = 1	Descriptor type is code segment:	If Code Segment (S = 1, E = 1)
	2	Conforming (C)	C = 1	Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.	
	1	Readable (R)	R = 0	Code segment may not be read.	
			R = 1	Code segment may be read.	
	0	Accessed (A)	A = 0	Segment has not been accessed.	
			A = 1	Segment selector has been loaded into segment register or used by selector test instructions.	

Code and data segments have several descriptor fields in common. The accessed **A** bit is set whenever the processor accesses a descriptor. The **A** bit is used by operating systems to keep usage statistics on a given segment. The **G** bit, or granularity bit, specifies if a segment length is byte-granular or page-granular. Intel486 Microprocessor segments can be one megabyte long with byte granularity (G=0) or four gigabytes with page granularity (G=1), (i.e., 2^{20} pages each page is 4 Kbytes in length). The granularity is totally unrelated to paging. A Intel486 Microprocessor system can consist of segments with byte granularity, and page granularity, whether or not paging is enabled.

The executable **E** bit tells if a segment is a code or data segment. A code segment (E=1, S=1) may be execute-only or execute/read as determined by the Read **R** bit. Code segments are execute only if R=0, and execute/read if R=1. Code segments may never be written into.

NOTE:
Code segments may be modified via aliases. Aliases are writeable data segments which occupy the same range of linear address space as the code segment.

The **D** bit indicates the default length for operands and effective addresses. If D=1 then 32-bit operands and 32-bit addressing modes are assumed. If D=0 then 16-bit operands and 16-bit addressing modes are assumed. Therefore all existing 80286 code segments will execute on the Intel486 Microprocessor assuming the D bit is set 0.

Another attribute of code segments is determined by the conforming **C** bit. Conforming segments, C=1, can be executed and shared by programs at different privilege levels. (See Section 4.4 **Protection**.)

Segments identified as data segments (E=0, S=1) are used for two types of Intel486 Microprocessor segments: stack and data segments. The expansion direction **(ED)** bit specifies if a segment expands downward (stack) or upward (data). If a segment is a stack segment all offsets must be greater than the segment limit. On a data segment all offsets must be less than or equal to the limit. In other words, stack segments start at the base linear address plus the maximum segment limit and grow down to the base linear address plus the limit. On the other hand, data segments start at the base linear address and expand to the base linear address plus limit.

The write **W** bit controls the ability to write into a segment. Data segments are read-only if W = 0. The stack segment must have W = 1.

The **B** bit controls the size of the stack pointer register. If B = 1, then PUSHes, POPs, and CALLs all use the 32-bit ESP register for stack references and assume an upper limit of FFFFFFFFH. If B = 0, stack instructions all use the 16-bit SP register and assume an upper limit of FFFFH.

4.3.4.3 System Descriptor Formats

System segments describe information about operating system tables, tasks, and gates. Figure 4.7 shows the general format of system segment descriptors, and the various types of system segments. Intel486 Microprocessor system descriptors contain a 32-bit base linear address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zero.

4.3.4.4 LDT Descriptors (S = 0, TYPE = 2)

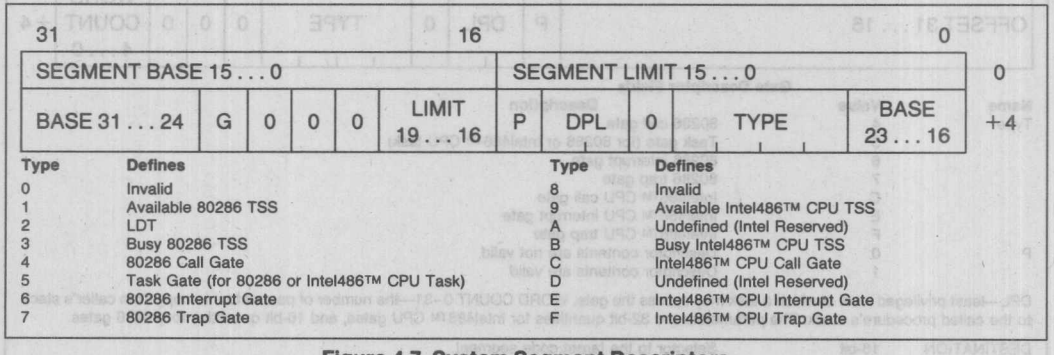
LDT descriptors (S = 0, TYPE = 2) contain information about Local Descriptor Tables. LDTs contain a table of segment descriptors, unique to a particular task. Since the instruction to load the LDTR is only available at privilege level 0, the DPL field is ignored. LDT descriptors are only allowed in the Global Descriptor Table (GDT).

4.3.4.5 TSS Descriptors (S = 0, TYPE = 1, 3, 9, B)

A Task State Segment (TSS) descriptor contains information about the location, size, and privilege level of a Task State Segment (TSS). A TSS in turn is a special fixed format segment which contains all the state information for a task and a linkage field to permit nesting tasks. The TYPE field is used to indicate whether the task is currently BUSY (i.e., on a chain of active tasks) or the TSS is available. The TYPE field also indicates if the segment contains a 80286 or an Intel486 Microprocessor TSS. The Task Register (TR) contains the selector which points to the current Task State Segment.

4.3.4.6 Gate Descriptors (S = 0, TYPE = 4–7, C, F)

Gates are used to control access to entry points within the target code segment. The various types of gate descriptors are **call** gates, **task** gates, **interrupt** gates, and **trap** gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the processor to automatically perform protection checks. It also allows system designers to control entry points to the operating system. Call gates are used to change privilege levels (see Section 4.4 **Protection**), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines.



Type	Defines	Type	Defines
0	Invalid	8	Invalid
1	Available 80286 TSS	9	Available Intel486™ CPU TSS
2	LDT	A	Undefined (Intel Reserved)
3	Busy 80286 TSS	B	Busy Intel486™ CPU TSS
4	80286 Call Gate	C	Intel486™ CPU Call Gate
5	Task Gate (for 80286 or Intel486™ CPU Task)	D	Undefined (Intel Reserved)
6	80286 Interrupt Gate	E	Intel486™ CPU Interrupt Gate
7	80286 Trap Gate	F	Intel486™ CPU Trap Gate

Figure 4.7. System Segment Descriptors

Figure 4.8 shows the format of the four types of gate descriptors. Call gates are primarily used to transfer program control to a more privileged level. The call gate descriptor consists of three fields: the access byte, a long pointer (selector and offset) which points to the start of a routine and a word count which specifies how many parameters are to be copied from the caller's stack to the stack of the called routine. The word count field is only used by call gates when there is a change in the privilege level, other types of gates ignore the word count field.

Interrupt and trap gates use the destination selector and destination offset fields of the gate descriptor as a pointer to the start of the interrupt or trap handler routines. The difference between interrupt gates and trap gates is that the interrupt gate disables interrupts (resets the IF bit) while the trap gate does not.

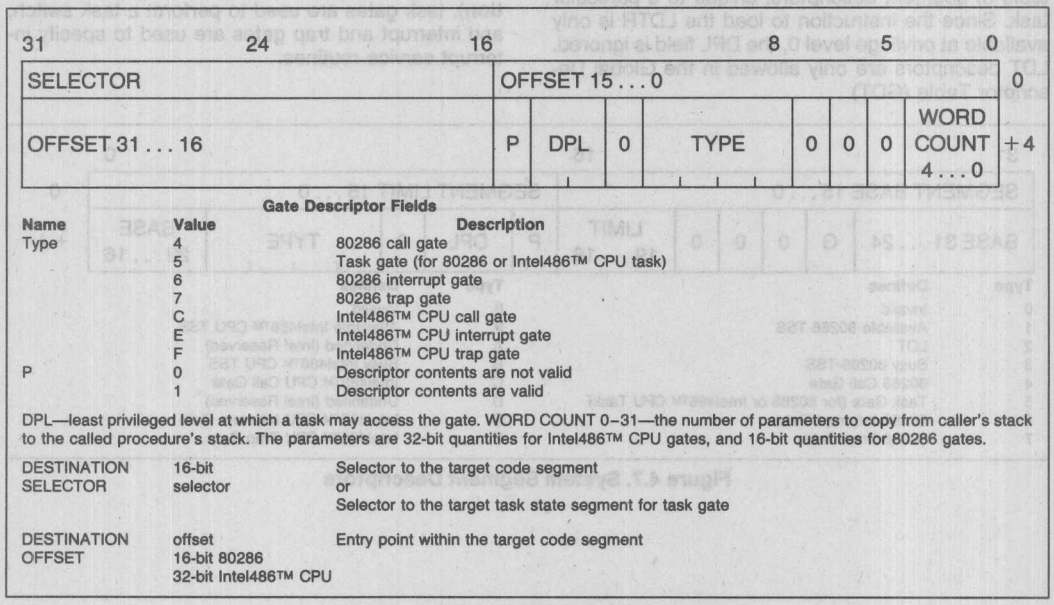
Task gates are used to switch tasks. Task gates may only refer to a task state segment (see Section 4.4.6 **Task Switching**) therefore only the destination selector portion of a task gate descriptor is used, and the destination offset is ignored.

Exception 13 is generated when a destination selector does not refer to a correct descriptor type, i.e., a code segment for an interrupt, trap or call gate, a TSS for a task gate.

The access byte format is the same for all gate descriptors. P = 1 indicates that the gate contents are valid. P = 0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (see Section 4.4 **Protection**). The S field, bit 4 of the access rights byte, must be 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 4.8.

4.3.4.7 Differences Between Intel486™ Microprocessor and 80286 Descriptors

In order to provide operating system compatibility between the 80286 and Intel486 Microprocessor, the Intel486 Microprocessor supports all of the 80286 segment descriptors. Figure 4.9 shows the general format of an 80286 system segment descriptor. The only differences between 80286 and Intel486 Microprocessor descriptor formats are that the values of the type fields, and the limit and base address fields have been expanded for the Intel486 Microprocessor. The 80286 system segment descriptors contained a 24-bit base address and 16-bit limit, while the Intel486 Microprocessor system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit.



Gate Descriptor Fields

Name	Value	Description
Type	4	80286 call gate
	5	Task gate (for 80286 or Intel486™ CPU task)
	6	80286 interrupt gate
	7	80286 trap gate
	C	Intel486™ CPU call gate
	E	Intel486™ CPU interrupt gate
	F	Intel486™ CPU trap gate
P	0	Descriptor contents are not valid
	1	Descriptor contents are valid

DPL—least privileged level at which a task may access the gate. WORD COUNT 0–31—the number of parameters to copy from caller's stack to the called procedure's stack. The parameters are 32-bit quantities for Intel486™ CPU gates, and 16-bit quantities for 80286 gates.

DESTINATION SELECTOR	16-bit selector	Selector to the target code segment or Selector to the target task state segment for task gate
DESTINATION OFFSET	offset 16-bit 80286 32-bit Intel486™ CPU	Entry point within the target code segment

Figure 4.8. Gate Descriptor Formats

By supporting 80286 system segments the Intel486 Microprocessor is able to execute 80286 application programs on an Intel486 Microprocessor operating system. This is possible because the processor automatically understands which descriptors are 80286-style descriptors and which descriptors are Intel486 Microprocessor-style descriptors. In particular, if the upper word of a descriptor is zero, then that descriptor is a 80286-style descriptor.

The only other differences between 80286-style descriptors and Intel486 Microprocessor descriptors is the interpretation of the word count field of call gates and the B bit. The word count field specifies the number of 16-bit quantities to copy for 80286 call gates and 32-bit quantities for Intel486 Microprocessor call gates. The B bit controls the size of PUSHes when using a call gate; if B = 0 PUSHes are 16 bits, if B = 1 PUSHes are 32 bits.

4.3.4.8 Selector Fields

A selector in Protected Mode has three fields: Local or Global Descriptor Table Indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 4.10. The TI bits select one of two memory-based tables of descriptors (the Global Descriptor Table or the Local Descriptor Table). The Index selects one of 8K descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

4.3.4.9 Segment Descriptor Cache

In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with that selector is automatically loaded (cached) on the chip. Once loaded, all references to that segment use the cached descriptor information instead of reaccessing the descriptor. The contents of the descriptor cache are not visible to the programmer. Since descriptor caches only change when a segment register is changed, programs which modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.

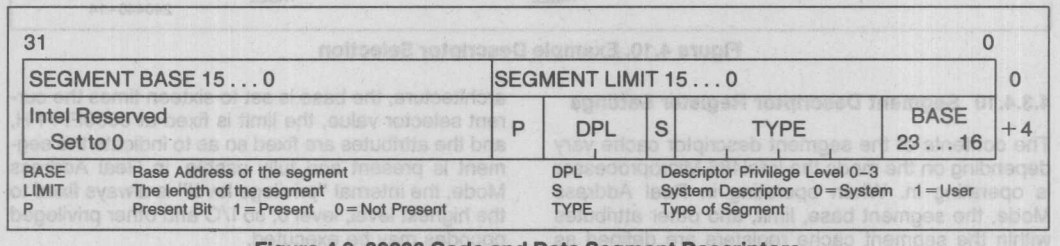


Figure 4.9. 80286 Code and Data Segment Descriptors

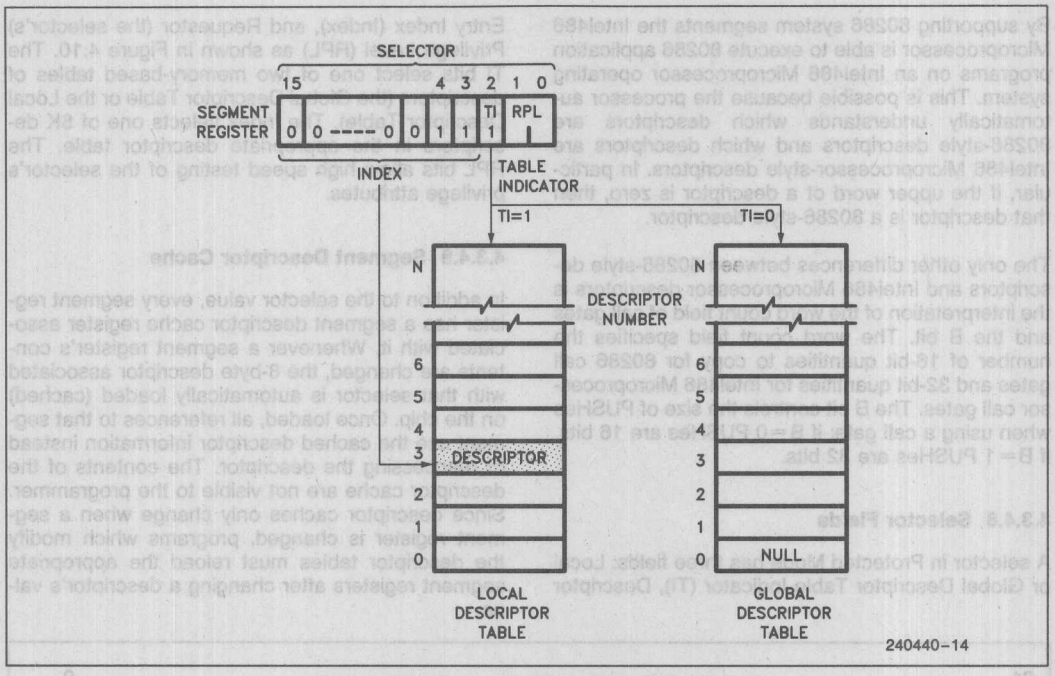
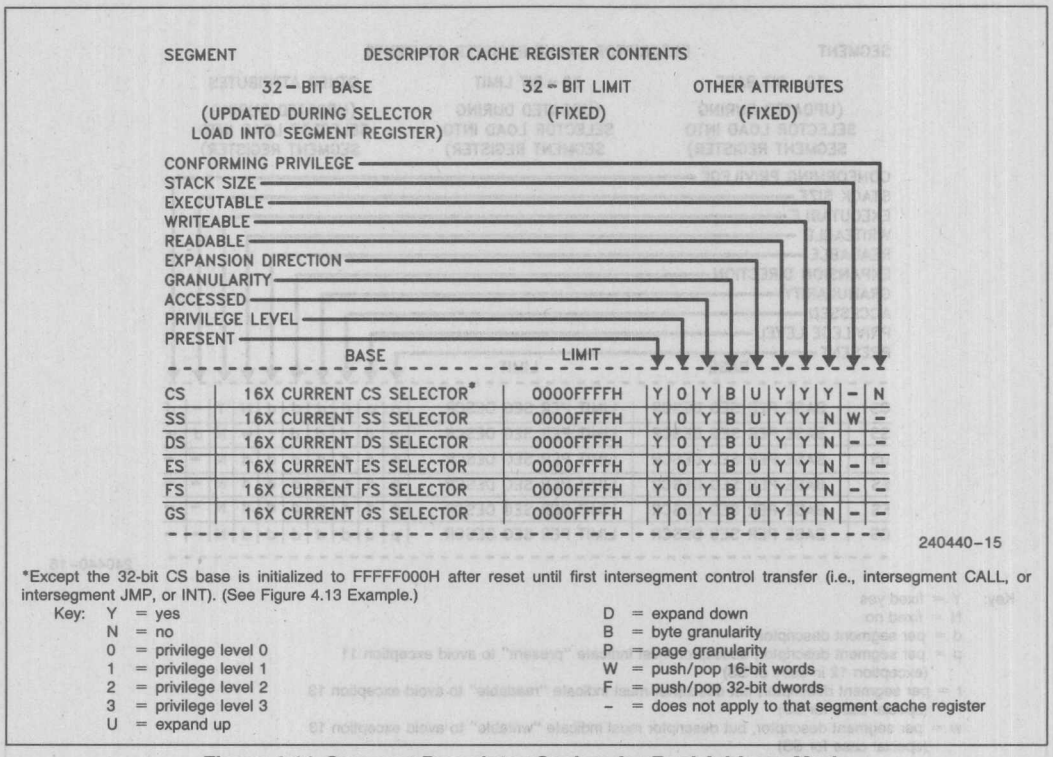


Figure 4.10. Example Descriptor Selection

4.3.4.10 Segment Descriptor Register Settings

The contents of the segment descriptor cache vary depending on the mode the Intel486 Microprocessor is operating in. When operating in Real Address Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.11. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. In Real Address Mode, the internal "privilege level" is always fixed to the highest level, level 0, so I/O and other privileged opcodes may be executed.

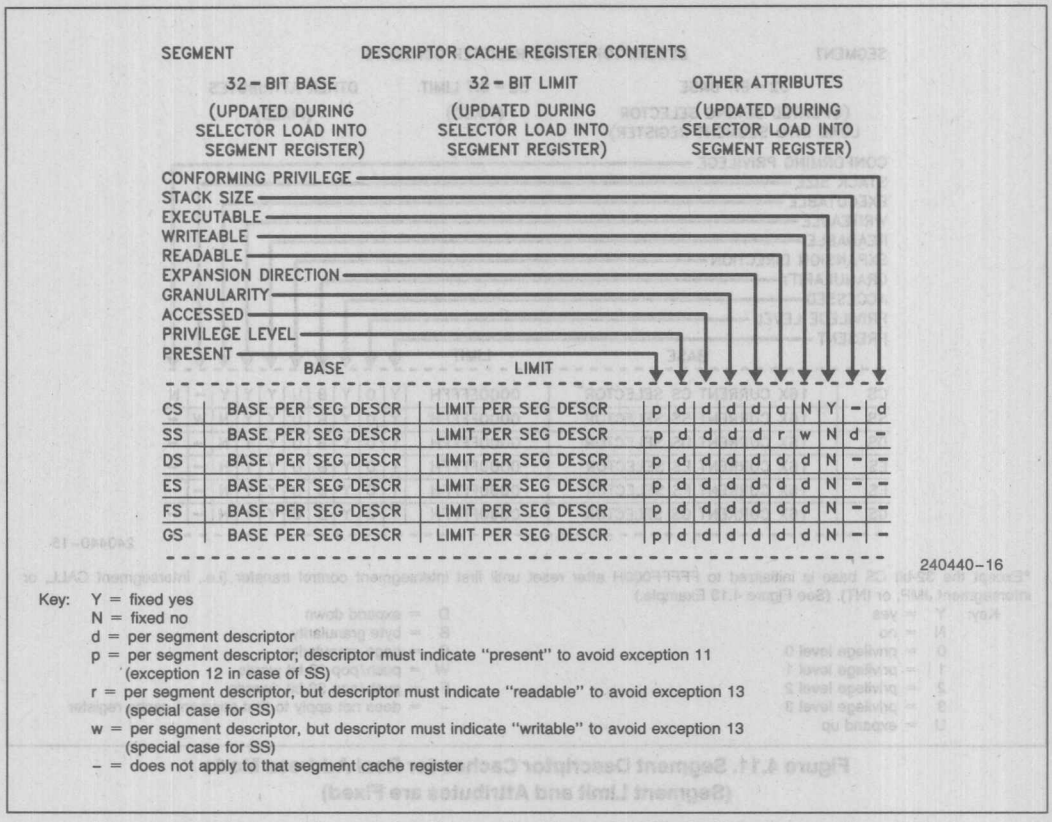


*Except the 32-bit CS base is initialized to FFFFF000H after reset until first intersegment control transfer (i.e., intersegment CALL, or intersegment JMP, or INT). (See Figure 4.13 Example.)

Key:
- Y = yes
- N = no
- 0 = privilege level 0
- 1 = privilege level 1
- 2 = privilege level 2
- 3 = privilege level 3
- U = expand up
- D = expand down
- B = byte granularity
- P = page granularity
- W = push/pop 16-bit words
- F = push/pop 32-bit dwords
- – = does not apply to that segment cache register

Figure 4.11. Segment Descriptor Caches for Real Address Mode (Segment Limit and Attributes are Fixed)

When operating in Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.12. In Protected Mode, each of these fields are defined according to the contents of the segment descriptor indexed by the selector value loaded into the segment register.



Key: Y = fixed yes
N = fixed no
d = per segment descriptor
p = per segment descriptor; descriptor must indicate "present" to avoid exception 11 (exception 12 in case of SS)
r = per segment descriptor, but descriptor must indicate "readable" to avoid exception 13 (special case for SS)
w = per segment descriptor, but descriptor must indicate "writable" to avoid exception 13 (special case for SS)
– = does not apply to that segment cache register

Figure 4.12. Segment Descriptor Caches for Protected Mode (Loaded per Descriptor)

When operating in a Virtual 8086 Mode within the Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.13. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. The virtual program executes at lowest privilege level, level 3, to allow trapping of all IOPL-sensitive instructions and level-0-only instructions.

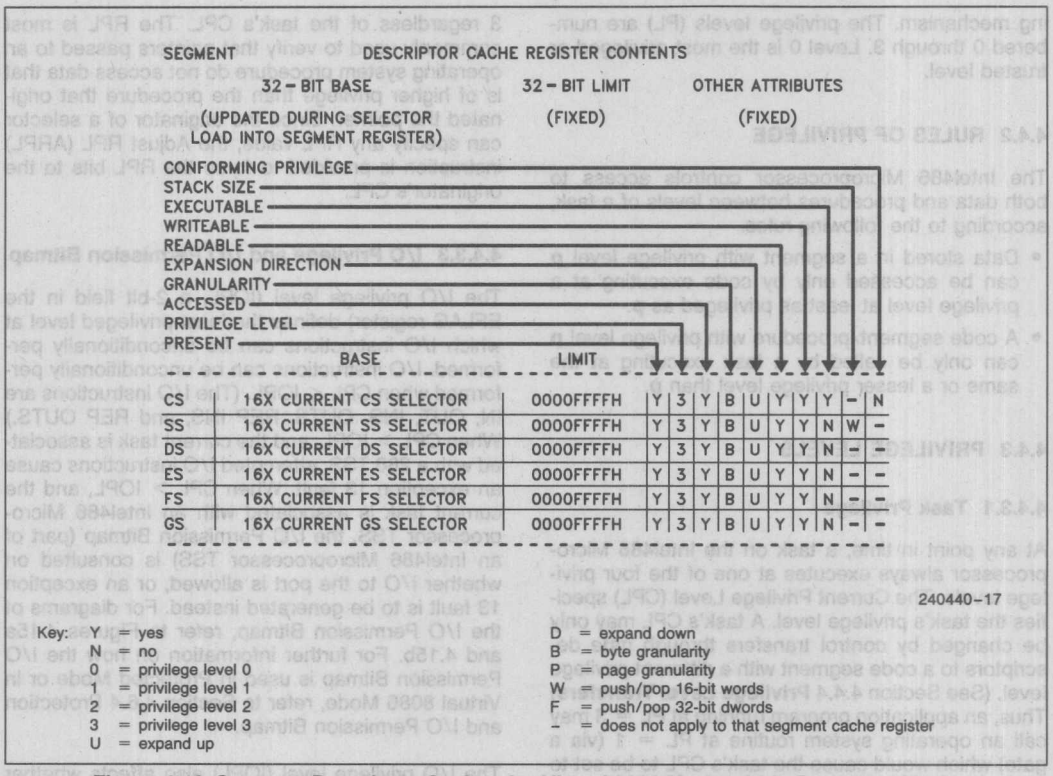


Figure 4.13. Segment Descriptor Caches for Virtual 8086 Mode within Protected Mode (Segment Limit and Attributes are Fixed)

4.4 Protection

4.4.1 PROTECTION CONCEPTS

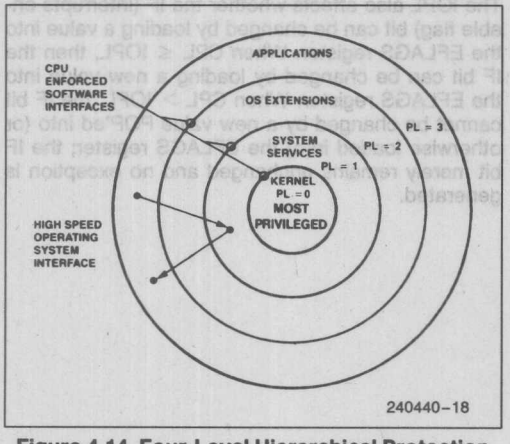


Figure 4.14. Four-Level Hierarchical Protection

The Intel486 Microprocessor has four levels of protection which are optimized to support the needs of a multi-tasking operating system to isolate and protect user programs from each other and the operating system. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. Unlike traditional microprocessor-based systems where this protection is achieved only through the use of complex external hardware and software the Intel486 Microprocessor provides the protection as part of its integrated Memory Management Unit. The Intel486 Microprocessor offers an additional type of protection on a page basis, when paging is enabled (See Section 4.5.3 **Page Level Protection**).

The four-level hierarchical privilege system is illustrated in Figure 4-14. It is an extension of the user/supervisor privilege mode commonly used by minicomputers and, in fact, the user/supervisor mode is fully supported by the Intel486 Microprocessor pag-

ing mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged or trusted level.

4.4.2 RULES OF PRIVILEGE

The Intel486 Microprocessor controls access to both data and procedures between levels of a task, according to the following rules.

- Data stored in a segment with privilege level **p** can be accessed only by code executing at a privilege level at least as privileged as **p**.
- A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p**.

4.4.3 PRIVILEGE LEVELS

4.4.3.1 Task Privilege

At any point in time, a task on the Intel486 Microprocessor always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies the task's privilege level. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level. (See Section 4.4.4 **Privilege Level Transfers**) Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

4.4.3.2 Selector Privilege (RPL)

The privilege level of a selector is specified by the RPL field. The RPL is the two least significant bits of the selector. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level for the use of a segment. This level is called the task's effective privilege level (EPL). The EPL is defined as being the least privileged (i.e. numerically larger) level of a task's CPL and a selector's RPL. Thus, if selector's RPL = 0 then the CPL always specifies the privilege level for making an access using the selector. On the other hand if RPL = 3 then a selector can only access segments at level 3 regardless of the task's CPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Since the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

4.4.3.3 I/O Privilege and I/O Permission Bitmap

The I/O privilege level (IOPL, a 2-bit field in the EFLAG register) defines the least privileged level at which I/O instructions can be unconditionally performed. I/O instructions can be unconditionally performed when CPL ≤ IOPL. (The I/O instructions are IN, OUT, INS, OUTS, REP INS, and REP OUTS.) When CPL > IOPL, and the current task is associated with a 286 TSS, attempted I/O instructions cause an exception 13 fault. When CPL > IOPL, and the current task is associated with an Intel486 Microprocessor TSS, the I/O Permission Bitmap (part of an Intel486 Microprocessor TSS) is consulted on whether I/O to the port is allowed, or an exception 13 fault is to be generated instead. For diagrams of the I/O Permission Bitmap, refer to Figures 4.15a and 4.15b. For further information on how the I/O Permission Bitmap is used in Protected Mode or in Virtual 8086 Mode, refer to Section 4.6.4 Protection and I/O Permission Bitmap.

The I/O privilege level (IOPL) also affects whether several other instructions can be executed or cause an exception 13 fault instead. These instructions are called "IOPL-sensitive" instructions and they are CLI and STI. (Note that the LOCK prefix is *not* IOPL-sensitive on the Intel486 Microprocessor.)

The IOPL also affects whether the IF (interrupts enable flag) bit can be changed by loading a value into the EFLAGS register. When CPL ≤ IOPL, then the IF bit can be changed by loading a new value into the EFLAGS register. When CPL > IOPL, the IF bit cannot be changed by a new value POP'ed into (or otherwise loaded into) the EFLAGS register; the IF bit merely remains unchanged and no exception is generated.

Table 4.2. Pointer Test Instructions

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

4.4.3.4 Privilege Validation

The Intel486 Microprocessor provides several instructions to speed pointer testing and help maintain system integrity by verifying that the selector value refers to an appropriate segment. Table 4.2 summarizes the selector validation procedures available for the Intel486 Microprocessor.

This pointer verification prevents the common problem of an application at PL = 3 calling a operating systems routine at PL = 0 and passing the operating system routine a "bad" pointer which corrupts a data structure belonging to the operating system. If the operating system routine uses the ARPL instruction to ensure that the RPL of the selector has no greater privilege than that of the caller, then this problem can be avoided.

4.4.3.5 Descriptor Access

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL as described above.

Any time an instruction loads data segment registers (DS, ES, FS, GS) the Intel486 Microprocessor makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segments or readable code segments. The data access rules are specified in Section 4.4.2 **Rules of Privilege**. The only exception to those rules is readable conforming code segments which can be accessed at any privilege level.

Finally the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL an exception 13 (general protection fault) is generated.

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load selectors into SS must refer to data segment descriptors for writeable data segments. The DPL and RPL must equal the CPL. All other descriptor types or a privilege level violation will cause exception 13. A stack not present fault causes exception 12. Note that an exception 11 is used for a not-present code or data segment.

4.4.4 PRIVILEGE LEVEL TRANSFERS

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 4.3. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only via control transfers, by using gates, task switches, and interrupt or trap gates.

Table 4.3. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
Task Switch	CALL, JMP	Task State Segment	GDT
	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag register) = 0
**NT (Nested Task bit of flag register) = 1

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13 (e.g. JMP through a call gate, or IRET from a normal subroutine call).

In order to provide further system security, all control transfers are also subject to the privilege rules.

The privilege rules require that:

— Privilege level transitions can only occur via gates.

— JMPs can be made to a non-conforming code segment with the same privilege or to a conforming code segment with greater or equal privilege.

— CALLs can be made to a non-conforming code segment with the same privilege or via a gate to a more privileged level.

— Interrupts handled within the task obey the same privilege rules as CALLs.

— Conforming Code segments are accessible by privilege levels which are the same or less privileged than the conforming-code segment's DPL.

— Both the requested privilege level (RPL) in the selector pointing to the gate and the task's CPL must be of equal or greater privilege than the gate's DPL.

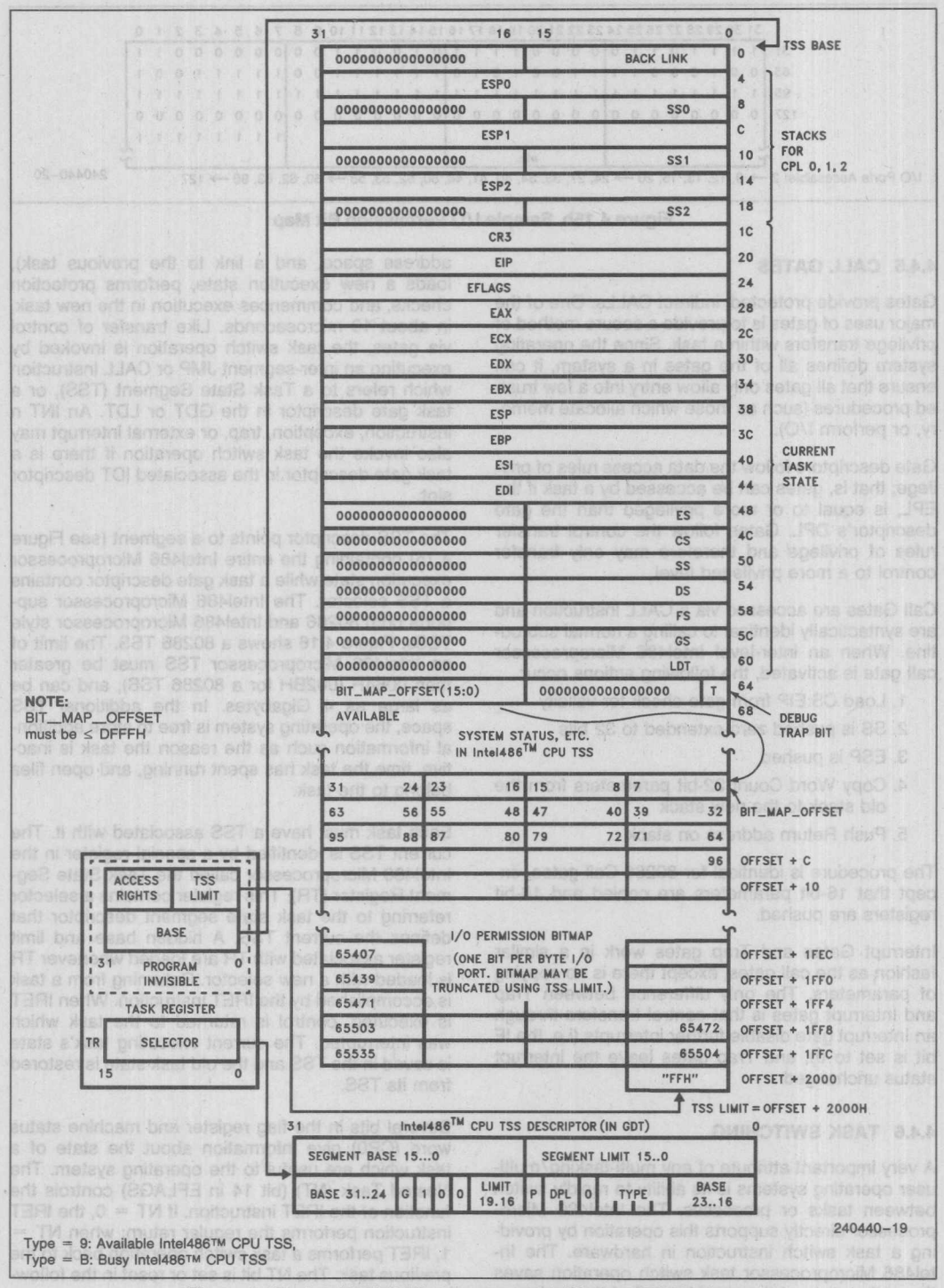
— The code segment selected in the gate must be the same or more privileged than the task's CPL.

— Return instructions that do not switch tasks can only return control to a code segment with same or less privilege.

— Task switches can be performed by a CALL, JMP, or INT which references either a task gate or task state segment who's DPL is less privileged or the same privilege as the old task's CPL.

Any control transfer that changes CPL within a task causes a change of stacks as a result of the privilege level change. The initial values of SS:ESP for privilege levels 0, 1, and 2 are retained in the task state segment (see Section 4.4.6 **Task Switching**). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and ESP registers and the previous stack pointer is pushed onto the new stack.

When RETurning to the original privilege level, use of the lower-privileged stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words (as specified in the gate's word count field) are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.



NOTE:
BIT_MAP_OFFSET must be ≤ DFFFH

Type = 9: Available Intel486™ CPU TSS,
Type = B: Busy Intel486™ CPU TSS

Figure 4.15a. Intel486™ Microprocessor TSS and TSS Registers

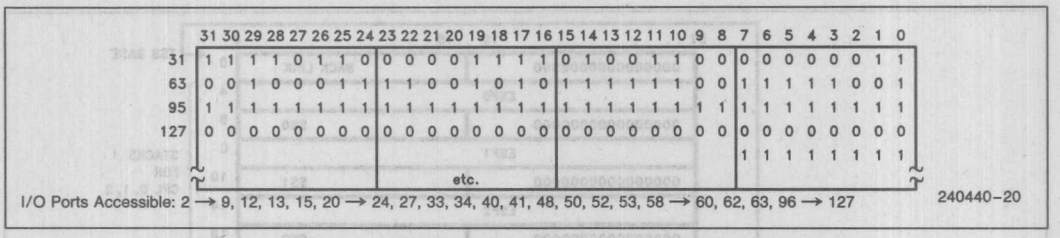


Figure 4.15b. Sample I/O Permission Bit Map

4.4.5 CALL GATES

Gates provide protected, indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures (such as those which allocate memory, or perform I/O).

Gate descriptors follow the data access rules of privilege; that is, gates can be accessed by a task if the EPL, is equal to or more privileged than the gate descriptor's DPL. Gates follow the control transfer rules of privilege and therefore may only transfer control to a more privileged level.

Call Gates are accessed via a CALL instruction and are syntactically identical to calling a normal subroutine. When an inter-level Intel486 Microprocessor call gate is activated, the following actions occur.

1. Load CS:EIP from gate check for validity
2. SS is pushed zero-extended to 32 bits
3. ESP is pushed
4. Copy Word Count 32-bit parameters from the old stack to the new stack
5. Push Return address on stack

The procedure is identical for 80286 Call gates, except that 16-bit parameters are copied and 16-bit registers are pushed.

Interrupt Gates and Trap gates work in a similar fashion as the call gates, except there is no copying of parameters. The only difference between Trap and Interrupt gates is that control transfers through an Interrupt gate disable further interrupts (i.e. the IF bit is set to 0), and Trap gates leave the interrupt status unchanged.

4.4.6 TASK SWITCHING

A very important attribute of any multi-tasking/multi-user operating systems is its ability to rapidly switch between tasks or processes. The Intel486 Microprocessor directly supports this operation by providing a task switch instruction in hardware. The Intel486 Microprocessor task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task, in about 10 microseconds. Like transfer of control via gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 4.15) containing the entire Intel486 Microprocessor execution state while a task gate descriptor contains a TSS selector. The Intel486 Microprocessor supports both 80286 and Intel486 Microprocessor style TSSs. Figure 4.16 shows a 80286 TSS. The limit of an Intel486 Microprocessor TSS must be greater than 0064H (002BH for a 80286 TSS), and can be as large as 4 Gigabytes. In the additional TSS space, the operating system is free to store additional information such as the reason the task is inactive, time the task has spent running, and open files belong to the task.

Each task must have a TSS associated with it. The current TSS is identified by a special register in the Intel486 Microprocessor called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The current executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word (CR0) give information about the state of a task which are useful to the operating system. The Nested Task (NT) (bit 14 in EFLAGS) controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular return; when NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

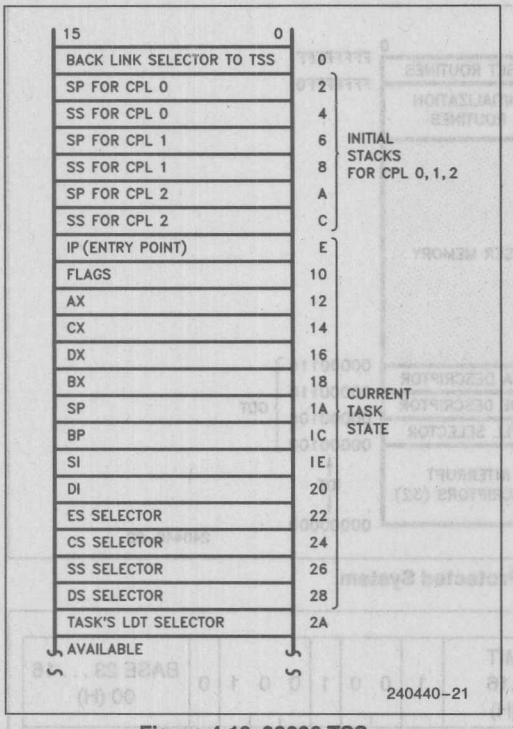


Figure 4.16. 80286 TSS

When a CALL or INT instruction initiates a task switch, the new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. (The NT bit will be restored after execution of the interrupt handler) NT may also be set or cleared by POPF or IRET instructions.

The Intel486 Microprocessor task state segment is marked busy by changing the descriptor type field from TYPE 9H to TYPE BH. An 80286 TSS is marked busy by changing the descriptor type field from TYPE 1 to TYPE 3. Use of a selector that references a busy task state segment causes an exception 13.

The Virtual Mode (VM) bit 17 is used to indicate if a task, is a virtual 8086 task. If VM = 1, then the tasks will use the Real Mode addressing mechanism. The virtual 8086 environment is only entered and exited via a task switch (see Section 4.6 **Virtual Mode**).

The FPU's state is not automatically saved when a task switch occurs, because the incoming task may not use the FPU. The Task Switched (TS) Bit (bit 3 in the CR0) helps deal with the FPU's state in a multitasking environment. Whenever the Intel486 Microprocessor switches tasks, it sets the TS bit. The Intel486 Microprocessor detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the FPU. A processor extension not present exception (7) will occur when attempting to execute a Floating Point or WAIT instruction if the Task Switched and Monitor coprocessor extension bits are both set (i.e. TS = 1 and MP = 1).

The **T** bit in the Intel486 Microprocessor TSS indicates that the processor should generate a debug exception when switching to a task. If T = 1 then upon entry to a new task a debug exception 1 will be generated.

4.4.7 INITIALIZATION AND TRANSITION TO PROTECTED MODE

Since the Intel486 Microprocessor begins executing in Real Mode immediately after RESET it is necessary to initialize the system tables and registers with the appropriate values.

The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256 bytes long, and GDT must contain descriptors for the initial code, and data segments. Figure 4.17 shows the tables and Figure 4.18 the descriptors needed for a simple Protected Mode Intel486 Microprocessor system. It has a single code and single data/stack segment each four gigabytes long and a single privilege level PL = 0.

The actual method of enabling Protected Mode is to load CR0 with the PE bit set, via the MOV CR0, R/M instruction. This puts the Intel486 Microprocessor in Protected Mode.

After enabling Protected Mode, the next instruction should execute an intersegment JMP to load the CS register and flush the instruction decode queue. The final step is to load all of the data segment registers with the initial selector values.

An alternate approach to entering Protected Mode which is especially appropriate for multi-tasking operating systems, is to use the built in task-switch to load all of the registers. In this case the GDT would contain two TSS descriptors in addition to the code and data descriptors needed for the first task. The first JMP instruction in Protected Mode would jump to the TSS causing a task switch and loading all of the registers with the values stored in the TSS. The Task State Segment Register should be initialized to point to a valid TSS descriptor since a task switch saves the state of the current task in a task state segment.

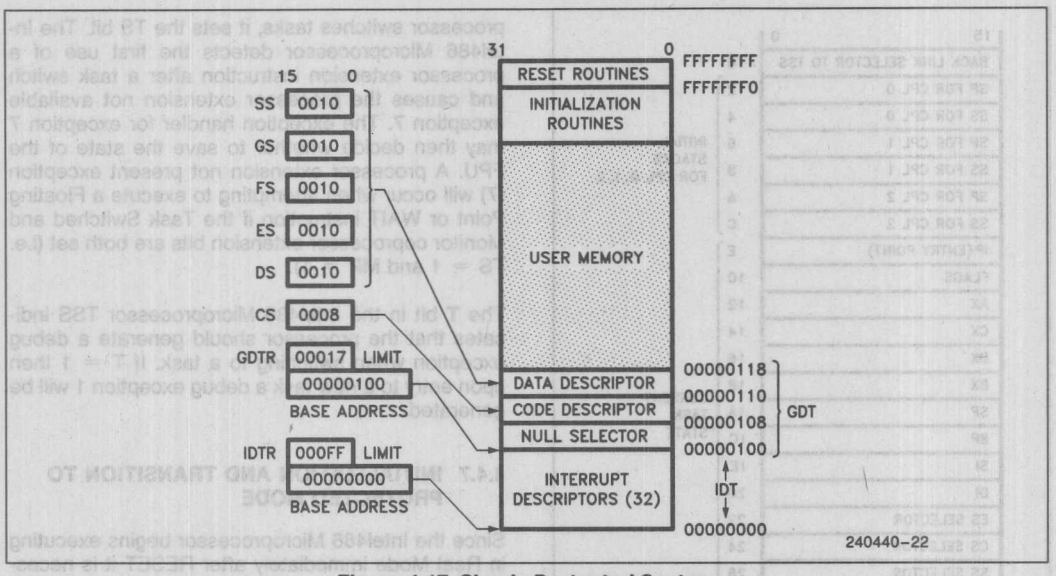


Figure 4.17. Simple Protected System

DATA DESCRIPTOR	2	BASE 31 . . . 24 00 (H)	G 1	D 1	0	0		LIMIT 19.16 F (H)	1	0	0	1	0	0	1	0	BASE 23 . . . 16 00 (H)
		SEGMENT BASE 15 . . . 0 0118 (H)							SEGMENT LIMIT 15 . . . 0 FFFF (H)								
CODE DESCRIPTOR	1	BASE 31 . . . 24 00 (H)	G 1	D 1	0	0		LIMIT 19.16 F (H)	1	0	0	1	1	0	1	0	BASE 23 . . . 16 00 (H)
		SEGMENT BASE 15 . . . 0 0118 (H)							SEGMENT LIMIT 15 . . . 0 FFFF (H)								
								NULL	DESCRIPTOR								
	0																
		31				24		16	15				8				0

Figure 4.18. GDT Descriptors for Simple System

4.4.8 TOOLS FOR BUILDING PROTECTED SYSTEMS

In order to simplify the design of a protected multitasking system, Intel provides a tool which allows the system designer an easy method of constructing the data structures needed for a Protected Mode Intel486 Microprocessor system. This tool is the builder BLD-386. BLD-386 lets the operating system writer specify all of the segment descriptors discussed in the previous sections (LDTs, IDTs, GDTs, Gates, and TSSs) in a high-level language.

4.5 Paging

4.5.1 PAGING CONCEPTS

Paging is another type of memory management useful for virtual memory multitasking operating systems. Unlike segmentation which modularizes programs and data into variable length segments, paging divides programs into multiple uniform size pages. Pages bear no direct relation to the logical

structure of a program. While segment selectors can be considered the logical "name" of a program module or data structure, a page most likely corresponds to only a portion of a module or data structure.

By taking advantage of the locality of reference displayed by most programs, only a small number of pages from each active task need be in memory at any one moment.

4.5.2 PAGING ORGANIZATION

4.5.2.1 Page Mechanism

The Intel486 Microprocessor uses two levels of tables to translate the linear address (from the segmentation unit) into a physical address. There are three components to the paging mechanism of the Intel486 Microprocessor: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the Intel486 Microprocessor paging mechanism are the same size, namely, 4 Kbytes. A uniform size for all of the elements simplifies memory allocation and reallocation schemes, since there is no problem with memory fragmentation. Figure 4.19 shows how the paging mechanism works.

4.5.2.2 Page Descriptor Base Register

CR2 is the Page Fault Linear Address register. It holds the 32-bit linear address which caused the last page fault detected.

CR3 is the Page Directory Physical Base Address Register. It contains the physical starting address of the Page Directory. The lower 12 bits of CR3 are always zero to ensure that the Page Directory is always page aligned. Loading it via a MOV CR3, reg instruction causes the Page Table Entry cache to be flushed, as will a task switch through a TSS which **changes** the value of CR0. (See 4.5.5 **Translation Lookaside Buffer**).

4.5.2.3 Page Directory

The Page Directory is 4 Kbytes long and allows up to 1024 Page Directory Entries. Each Page Directory Entry contains the address of the next level of tables, the Page Tables and information about the page table. The contents of a Page Directory Entry are shown in Figure 4.20. The upper 10 bits of the linear address (A22–A31) are used as an index to select the correct Page Directory Entry.

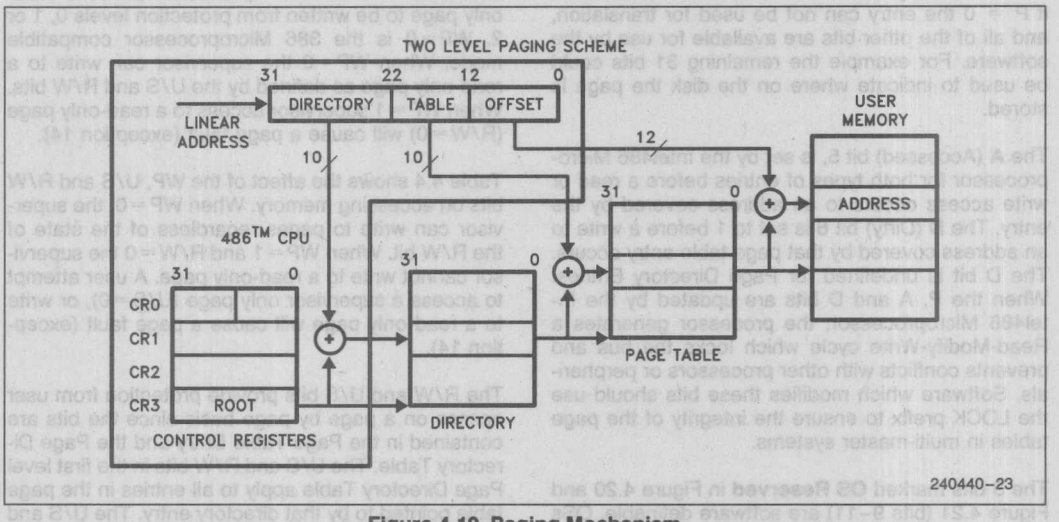


Figure 4.19. Paging Mechanism

31 12	11 10	9	8	7	6	5	4	3	2	1	0
PAGE TABLE ADDRESS 31..12	OS RESERVED		0	0	D	A	PCD	PWT	U/S	R/W	P

Figure 4.20. Page Directory Entry (Points to Page Table)

31 12	11 10	9	8	7	6	5	4	3	2	1	0
PAGE FRAME ADDRESS 31..12	OS RESERVED		0	0	D	A	PCD	PWT	U/S	R/W	P

Figure 4.21. Page Table Entry (Points to Page)

4.5.2.4 Page Tables

Each Page Table is 4 Kbytes and holds up to 1024 Page Table Entries. Page Table Entries contain the starting address of the page frame and statistical information about the page (see Figure 4.21). Address bits A12–A21 are used as an index to select one of the 1024 Page Table Entries. The 20 upper-bit page frame address is concatenated with the lower 12 bits of the linear address to form the physical address. Page tables can be shared between tasks and swapped to disks.

4.5.2.5 Page Directory/Table Entries

The lower 12 bits of the Page Table Entries and Page Directory Entries contain statistical information about pages and page tables respectively. The **P** (Present) bit 0 indicates if a Page Directory or Page Table entry can be used in address translation. If P = 1 the entry can be used for address translation if P = 0 the entry can not be used for translation, and all of the other bits are available for use by the software. For example the remaining 31 bits could be used to indicate where on the disk the page is stored.

The **A** (Accessed) bit 5, is set by the Intel486 Microprocessor for both types of entries before a read or write access occurs to an address covered by the entry. The **D** (Dirty) bit 6 is set to 1 before a write to an address covered by that page table entry occurs. The D bit is undefined for Page Directory Entries. When the P, A and D bits are updated by the Intel486 Microprocessor, the processor generates a Read-Modify-Write cycle which locks the bus and prevents conflicts with other processors or perpherials. Software which modifies these bits should use the LOCK prefix to ensure the integrity of the page tables in multi-master systems.

The 3 bits marked **OS Reserved** in Figure 4.20 and Figure 4.21 (bits 9–11) are software definable. OSs are free to use these bits for whatever purpose they wish. An example use of the **OS Reserved** bits would be to store information about page aging. By keeping track of how long a page has been in memory since being accessed, an operating system can implement a page replacement algorithm like Least Recently Used.

The (User/Supervisor) U/S bit 2 and the (Read/Write) R/W bit 1 are used to provide protection attributes for individual pages.

4.5.3 PAGE LEVEL PROTECTION (R/W, U/S BITS)

The Intel486 Microprocessor provides a set of protection attributes for paging systems. The paging mechanism distinguishes between two levels of protection: User which corresponds to level 3 of the segmentation based protection, and supervisor which encompasses all of the other protection levels (0, 1, 2).

The R/W and U/S bits are used in conjunction with the WP bit in the flags register (EFLAGS). The 386 Microprocessor does not contain the WP bit. The WP bit has been added to the Intel486 Microprocessor to protect read-only pages from supervisor write accesses. The 386 Microprocessor allows a read-only page to be written from protection levels 0, 1 or 2. WP=0 is the 386 Microprocessor compatible mode. When WP=0 the supervisor can write to a read-only page as defined by the U/S and R/W bits. When WP=1 supervisor access to a read-only page (R/W=0) will cause a page fault (exception 14).

Table 4.4 shows the affect of the WP, U/S and R/W bits on accessing memory. When WP=0, the supervisor can write to pages regardless of the state of the R/W bit. When WP=1 and R/W=0 the supervisor cannot write to a read-only page. A user attempt to access a supervisor only page (U/S=0), or write to a read only page will cause a page fault (exception 14).

The R/W and U/S bits provide protection from user access on a page by page basis since the bits are contained in the Page Table Entry and the Page Directory Table. The U/S and R/W bits in the first level Page Directory Table apply to all entries in the page table pointed to by that directory entry. The U/S and R/W bits in the second level Page Table Entry apply only to the page described by that entry. The most restrictive of the U/S and R/W bits from the Page Directory Table and the Page Table Entry are used to address a page.

Example: If the U/S and R/W bits for the Page Directory entry were 10 (user read/execute) and the

U/S and R/W bits for the Page Table Entry were 01 (no user access at all), the access rights for the page would be 01, the numerically smaller of the two.

Note that a given segment can be easily made read-only for level 0, 1 or 2 via use of segmented protection mechanisms. (Section 4.4 **Protection**).

4.5.4 PAGE CACHEABILITY (PWT AND PCD BITS)

PWT (page write through) and PCD (page cache disable) are two new bits defined in entries in both levels of the page table structure, the Page Directory Table and the Page Table Entry. PCD and PWT control page cacheability and write policy.

PWT controls write policy. PWT = 1 defines a write-through policy for the current page. PWT = 0 allows the possibility of write-back. PWT is ignored internally because the Intel486 microprocessor has a write-through cache. PWT can be used to control the write policy of a second level cache.

PCD controls cacheability. PCD = 0 enables caching in the on-chip cache. PCD alone does not enable caching, it must be conditioned by the KEN# (cache enable) input signal and the state of the CD (cache disable bit) and NW (no write-through) bits in control register 0 (CR0). When PCD = 1, caching is disabled regardless of the state of KEN#, CD and NW. (See Section 5.0, **On-Chip Cache**).

The state of the PCD and PWT bits are driven out on the PCD and PWT pins during a memory access.

The PWT and PCD bits for a bus cycle are obtained either from control register 3 (CR3), the Page Directory Entry or the Page Table Entry, depending on the type of cycle run. However, when paging is disabled (PG = 0 in CR0) or for cycles which bypass paging (i.e., I/O (input/output) references, INTR (interrupt request) and HALT cycles), the PCD and PWT bits of CR3 are ignored. The Intel486 CPU assumes PCD = 0 and PWT = 0 and drives these values on the PCD and PWT pins.

When paging is enabled (PG = 1 in CR0), the bits from the page table entry are cached in the translation lookaside buffer (TLB), and are driven any time the page mapped by the TLB entry is referenced. For normal memory cycles run with paging enabled, the PWT and PCD bits are taken from the Page Table Entry. During TLB refresh cycles when the Page Directory and Page Table entries are read, the PWT and PCD bits must be obtained elsewhere. The bits are taken from CR3 when a Page Directory Entry is being read. The bits are taken from the Page Directory Entry when the Page Table Entry is being updated.

The PCD or PWT bits in CR3 are initialized to zero at reset, but can be set to any value by level 0 software.

4.5.5 TRANSLATION LOOKASIDE BUFFER

The Intel486 Microprocessor paging hardware is designed to support demand paged virtual memory systems. However, performance would degrade substantially if the processor was required to access two levels of tables for every memory reference. To solve this problem, the Intel486 Microprocessor keeps a cache of the most recently accessed pages, this cache is called the Translation Lookaside Buffer (TLB). The TLB is a four-way set associative 32-entry page table cache. It automatically keeps the most commonly used Page Table Entries in the processor. The 32-entry TLB coupled with a 4K page size, results in coverage of 128 Kbytes of memory addresses. For many common multi-tasking systems, the TLB will have a hit rate of about 98%. This means that the processor will only have to access the two-level page structure on 2% of all memory references. Figure 4.22 illustrates how the TLB complements the Intel486 Microprocessor's paging mechanism.

Reading a new entry into the TLB (TLB refresh) is a two step process handled by the Intel486 microprocessor hardware. The sequence of data cycles to perform a TLB refresh are:

Table 4.4. Page Level Protection Attributes

U/S	R/W	WP	User Access	Supervisor Access
0	0	0	None	Read/Write/Execute
0	1	0	None	Read/Write/Execute
1	0	0	Read/Execute	Read/Write/Execute
1	1	0	Read/Write/Execute	Read/Write/Execute
0	0	1	None	Read/Execute
0	1	1	None	Read/Write/Execute
1	0	1	Read/Execute	Read/Execute
1	1	1	Read/Write/Execute	Read/Write/Execute

1. Read the correct Page Directory Entry, as pointed to by the page base register and the upper 10 bits of the linear address. The page base register is in control register 3.

1a. Optionally perform a locked read/write to set the accessed bit in the directory entry. The directory entry will actually get read twice if the Intel486 Microprocessor needs to set any of the bits in the entry. If the page directory entry changes between the first and second reads, the data returned for the second read will be used.

2. Read the correct entry in the Page Table and place the entry in the TLB.

2a. Optionally perform a locked read/write to set the accessed and/or dirty bit in the page table entry. Again, note that the page table entry will actually get read twice if the Intel486 Microprocessor needs to set any of the bits in the entry. Like the directory entry, if the data changes between the first and second read the data returned for the second read will be used.

Note that the directory entry must always be read into the processor, since directory entries are never placed in the paging TLB. Page faults can be signaled from either the page directory read or the page table read. Page directory and page table entries may be placed in the Intel486 on-chip cache just like normal data.

4.5.6 PAGING OPERATION

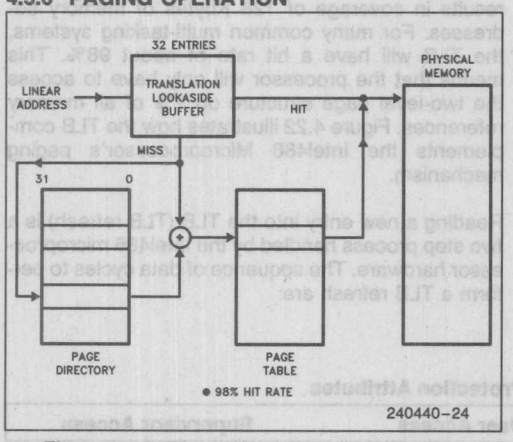


Figure 4.22. Translation Lookaside Buffer

The paging hardware operates in the following fashion. The paging unit hardware receives a 32-bit linear address from the segmentation unit. The upper 20 linear address bits are compared with all 32 entries in the TLB to determine if there is a match. If there is a match (i.e., a TLB hit), then the 32-bit physical address is calculated and will be placed on the address bus.

However, if the page table entry is not in the TLB, the Intel486 Microprocessor will read the appropriate Page Directory Entry. If P = 1 on the Page Directory Entry indicating that the page table is in memory, then the Intel486 Microprocessor will read the appropriate Page Table Entry and set the Access bit. If P = 1 on the Page Table Entry indicating that the page is in memory, the Intel486 Microprocessor will update the Access and Dirty bits as needed and fetch the operand. The upper 20 bits of the linear address, read from the page table, will be stored in the TLB for future accesses. However, if P = 0 for either the Page Directory Entry or the Page Table Entry, then the processor will generate a page fault, an Exception 14.

The processor will also generate an exception 14 page fault, if the memory reference violated the page protection attributes (i.e., U/S or R/W) (e.g., trying to write to a read-only page). CR2 will hold the linear address which caused the page fault. If a second page fault occurs, while the processor is attempting to enter the service routine for the first, then the processor will invoke the page fault (exception 14) handler a second time, rather than the double fault (exception 8) handler. Since Exception 14 is classified as a fault, CS: EIP will point to the instruction causing the page fault. The 16-bit error code pushed as part of the page fault handler will contain status bits which indicate the cause of the page fault.

The 16-bit error code is used by the operating system to determine how to handle the page fault. Figure 4.23a shows the format of the page-fault error code and the interpretation of the bits.

NOTE:
Even though the bits in the error code (U/S, W/R, and P) have similar names as the bits in the Page Directory/Table Entries, the interpretation of the error code bits is different. Figure 4.23b indicates what type of access caused the page fault.

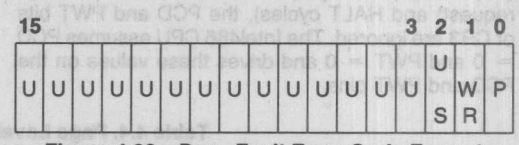


Figure 4.23a. Page Fault Error Code Format

U/S: The U/S bit indicates whether the access causing the fault occurred when the processor was executing in User Mode (U/S = 1) or in Supervisor mode (U/S = 0).

W/R: The W/R bit indicates whether the access causing the fault was a Read (W/R = 0) or a Write (W/R = 1).

P: The P bit indicates whether a page fault was caused by a not-present page (P = 0), or by a page level protection violation (P = 1).

U: UNDEFINED

U/S	W/R	Access Type
0	0	Supervisor* Read
0	1	Supervisor Write
1	0	User Read
1	1	User Write

*Descriptor table access will fault with U/S = 0, even if the program is executing at level 3.

Figure 4.23b. Type of Access Causing Page Fault

4.5.7 OPERATING SYSTEM RESPONSIBILITIES

The Intel486 Microprocessor takes care of the page address translation process, relieving the burden from an operating system in a demand-paged system. The operating system is responsible for setting up the initial page tables, and handling any page faults. The operating system also is required to invalidate (i.e., flush) the TLB when any changes are made to any of the page table entries. The operating system must reload CR3 to cause the TLB to be flushed.

Setting up the tables is simply a matter of loading CR3 with the address of the Page Directory, and allocating space for the Page Directory and the Page Tables. The primary responsibility of the operating system is to implement a swapping policy and handle all of the page faults.

A final concern of the operating system is to ensure that the TLB cache matches the information in the paging tables. In particular, any time the operating system sets the P present bit of page table entry to zero, the TLB must be flushed. Operating systems may want to take advantage of the fact that CR3 is stored as part of a TSS, to give every task or group of tasks its own set of page tables.

4.6 Virtual 8086 Environment

4.6.1 EXECUTING 8086 PROGRAMS

The Intel486 Microprocessor allows the execution of 8086 application programs in both Real Mode and in the Virtual 8086 Mode (Virtual Mode). Of the two methods, Virtual 8086 Mode offers the system designer the most flexibility. The Virtual 8086 Mode allows the execution of 8086 applications, while still allowing the system designer to take full advantage of the Intel486 Microprocessor protection mechanism. In particular, the Intel486 Microprocessor allows the simultaneous execution of 8086 operating systems and its applications, and an Intel486 Microprocessor operating system and both 80286 and Intel486 Microprocessor applications. Thus, in a multiuser Intel486 Microprocessor computer, one person could be running an MS-DOS spreadsheet, another person using MS-DOS, and a third person could be running multiple Unix utilities and applications. Each person in this scenario would believe that he had the computer completely to himself. Figure 4.24 illustrates this concept.

4.6.2 VIRTUAL 8086 MODE ADDRESSING MECHANISM

One of the major differences between Intel486 Microprocessor Real and Protected modes is how the segment selectors are interpreted. When the processor is executing in Virtual 8086 Mode the segment registers are used in an identical fashion to Real Mode. The contents of the segment register is shifted left 4 bits and added to the offset to form the segment base linear address.

The Intel486 Microprocessor allows the operating system to specify which programs use the 8086 style address mechanism, and which programs use Protected Mode addressing, on a per task basis. Through the use of paging, the one megabyte address space of the Virtual Mode task can be mapped to anywhere in the 4 gigabyte linear address space of the Intel486 Microprocessor. Like Real Mode, Virtual Mode effective addresses (i.e., segment offsets) that exceed 64 Kbyte will cause an exception 13. However, these restrictions should not prove to be important, because most tasks running in Virtual 8086 Mode will simply be existing 8086 application programs.

4.6.3 PAGING IN VIRTUAL MODE

The paging hardware allows the concurrent running of multiple Virtual Mode tasks, and provides protection and operating system isolation. Although it is not strictly necessary to have the paging hardware enabled to run Virtual Mode tasks, it is needed in order to run multiple Virtual Mode tasks or to relocate the address space of a Virtual Mode task to physical address space greater than one megabyte.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into up to 256 pages. Each one of the pages can be located anywhere within the maximum 4 gigabyte physical address space of the Intel486 Microprocessor. In addition, since CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations.

Finally, the paging hardware allows the sharing of the 8086 operating system code between multiple 8086 applications. Figure 4.24 shows how the Intel486 Microprocessor paging hardware enables multiple 8086 programs to run under a virtual memory demand paged system.

4.6.4 PROTECTION AND I/O PERMISSION BITMAP

All Virtual 8086 Mode programs execute at privilege level 3, the level of least privilege. As such, Virtual 8086 Mode programs are subject to all of the protection checks defined in Protected Mode. (This is different from Real Mode which implicitly is executing at privilege level 0, the level of greatest privilege.) Thus, an attempt to execute a privileged instruction when in Virtual 8086 Mode will cause an exception 13 fault.

The following are privileged instructions, which may be executed only at Privilege Level 0. Therefore, attempting to execute these instructions in Virtual 8086 Mode (or anytime CPL > 0) causes an exception 13 fault:

```
LIDT;  MOV DRn,reg;  MOV reg,DRn;
LGDT;  MOV TRn,reg;  MOV reg,TRn;
LMSW;  MOV CRn,reg;  MOV reg,CRn.
CLTS;
HLT;
```

Several instructions, particularly those applying to the multitasking model and protection model, are available only in Protected Mode. Therefore, attempting to execute the following instructions in Real Mode or in Virtual 8086 Mode generates an exception 6 fault:

```
LTR;   STR;
LLDT;  SLDT;
LAR;   VERR;
LSL;   VERW;
ARPL.
```

The instructions which are IOPL-sensitive in Protected Mode are:

```
IN;        STI;
OUT;       CLI
INS;
OUTS;
REP INS;
REP OUTS;
```

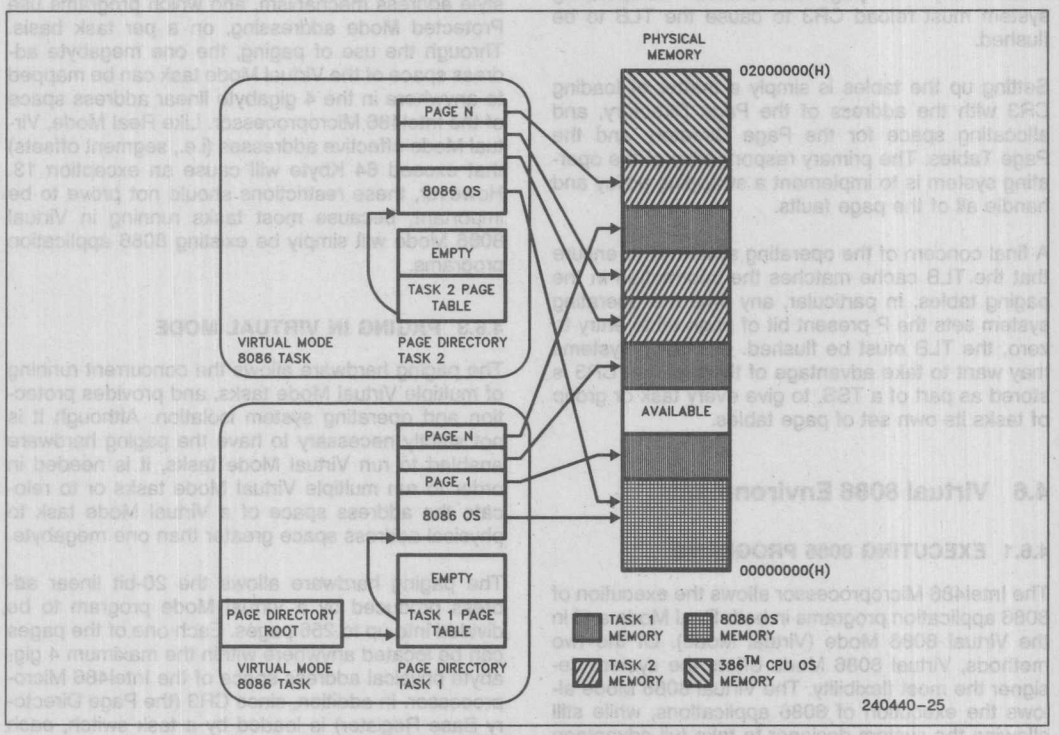


Figure 4.24. Virtual 8086 Environment Memory Management

In Virtual 8086 Mode, a slightly different set of instructions are made IOPL-sensitive. The following instructions are IOPL-sensitive in Virtual 8086 Mode:

```
INT n;    STI;
PUSHF;    CLI;
POPF;     IRET
```

The PUSHF, POPF, and IRET instructions are IOPL-sensitive in Virtual 8086 Mode only. This provision allows the IF flag (interrupt enable flag) to be virtualized to the Virtual 8086 Mode program. The INT n software interrupt instruction is also IOPL-sensitive in Virtual 8086 Mode. Note, however, that the INT 3 (opcode 0CCH), INTO, and BOUND instructions are not IOPL-sensitive in Virtual 8086 mode (they aren't IOPL sensitive in Protected Mode either).

Note that the I/O instructions (IN, OUT, INS, OUTS, REP INS, and REP OUTS) are **not** IOPL-sensitive in Virtual 8086 mode. Rather, the I/O instructions become automatically sensitive to the **I/O Permission Bitmap** contained in the **Intel486 Microprocessor Task State Segment**. The I/O Permission Bitmap, automatically used by the Intel486 Microprocessor in Virtual 8086 Mode, is illustrated by Figures 4.15a and 4.15b.

The I/O Permission Bitmap can be viewed as a 0–64 Kbit bit string, which begins in memory at offset Bit_Map_Offset in the current TSS. Bit_Map_Offset must be ≤ DFFFH so the entire bit map and the byte FFH which follows the bit map are all at offsets ≤ FFFFH from the TSS base. The 16-bit pointer Bit_Map_Offset (15:0) is found in the word beginning at offset 66H (102 decimal) from the TSS base, as shown in Figure 4.15a.

Each bit in the I/O Permission Bitmap corresponds to a single byte-wide I/O port, as illustrated in Figure 4.15a. If a bit is 0, I/O to the corresponding byte-wide port can occur without generating an exception. Otherwise the I/O instruction causes an exception 13 fault. Since every byte-wide I/O port must be protectable, all bits corresponding to a word-wide or dword-wide port must be 0 for the word-wide or dword-wide I/O to be permitted. If all the referenced bits are 0, the I/O will be allowed. If any referenced bits are 1, the attempted I/O will cause an exception 13 fault.

Due to the use of a pointer to the base of the I/O Permission Bitmap, the bitmap may be located anywhere within the TSS, or may be ignored completely by pointing the Bit_Map_Offset (15:0) beyond the limit of the TSS segment. In the same manner, only a small portion of the 64K I/O space need have an associated map bit, by adjusting the TSS limit to truncate the bitmap. This eliminates the commitment of 8K of memory when a complete bitmap is not required, while allowing the fully general case if desired.

EXAMPLE OF BITMAP FOR I/O PORTS 0–255: Setting the TSS limit to {bit_Map_Offset + 31 +1**} [** see note below] will allow a 32-byte bitmap for the I/O ports #0–255, plus a terminator byte of all 1's [** see note below]. This allows the I/O bitmap to control I/O Permission to I/O port 0–255 while causing an exception 13 fault on attempted I/O to any I/O port 80256 through 65,565.

****IMPORTANT IMPLEMENTATION NOTE:** Beyond the last byte of I/O mapping information in the I/O Permission Bitmap **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the Intel486 Microprocessor TSS segment (see Figure 4.15a).

4.6.5 INTERRUPT HANDLING

In order to fully support the emulation of an 8086 machine, interrupts in Virtual 8086 Mode are handled in a unique fashion. When running in Virtual Mode all interrupts and exceptions involve a privilege change back to the host Intel486 Microprocessor operating system. The Intel486 Microprocessor operating system determines if the interrupt comes from a Protected Mode application or from a Virtual Mode program by examining the VM bit in the EFLAGS image stored on the stack.

When a Virtual Mode program is interrupted and execution passes to the interrupt routine at level 0, the VM bit is cleared. However, the VM bit is still set in the EFLAG image on the stack.

The Intel486 Microprocessor operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The Intel486 Microprocessor operating system may choose to let the 8086 operating system handle the interrupt or it may emulate the function of the interrupt handler. For example, many 8086 operating system calls are accessed by PUSHing parameters on the stack, and then executing an INT n instruction. If the IOPL is set to 0 then all INT n instructions will be intercepted by the Intel486 Microprocessor operating system. The Intel486 Microprocessor operating system could emulate the 8086 operating system's call. Figure 4.25 shows how the Intel486 Microprocessor operating system could intercept an 8086 operating system's call to "Open a File".

An Intel486 Microprocessor operating system can provide a Virtual 8086 Environment which is totally transparent to the application software via intercepting and then emulating 8086 operating system's calls, and intercepting IN and OUT instructions.

4.6.6 ENTERING AND LEAVING VIRTUAL 8086 MODE

Virtual 8086 mode is entered by executing an IRET instruction (at CPL = 0), or Task Switch (at any CPL) to an Intel486 Microprocessor task whose Intel486 Microprocessor TSS has a FLAGS image containing a 1 in the VM bit position while the processor is executing in Protected Mode. That is, one way to enter Virtual 8086 mode is to switch to a task with an Intel486 Microprocessor TSS that has a 1 in the VM bit in the EFLAGS image. The other way is to execute a 32-bit IRET instruction at privilege level 0, where the stack has a 1 in the VM bit in the EFLAGS image. POPF does not affect the VM bit, even if the processor is in Protected Mode or level 0, and so cannot be used to enter Virtual 8086 Mode. PUSHF always pushes a 0 in the VM bit, even if the processor is in Virtual 8086 Mode, so that a program cannot tell if it is executing in REAL mode, or in Virtual 8086 mode.

The VM bit can be set by executing an IRET instruction only at privilege level 0, or by any instruction or Interrupt which causes a task switch in Protected Mode (with VM = 1 in the new FLAGS image), and can be cleared only by an interrupt or exception in Virtual 8086 Mode. IRET and POPF instructions executed in REAL mode or Virtual 8086 mode will not change the value in the VM bit.

The transition out of virtual 8086 mode to Intel486 Microprocessor protected mode occurs only on receipt of an interrupt or exception (such as due to a sensitive instruction). In Virtual 8086 mode, all interrupts and exceptions vector through the protected mode IDT, and enter an interrupt handler in protected Intel486 Microprocessor mode. That is, as part of interrupt processing, the VM bit is cleared.

Because the matching IRET must occur from level 0, if an Interrupt or Trap Gate is used to field an interrupt or exception out of Virtual 8086 mode, the Gate must perform an inter-level interrupt only to level 0. Interrupt or Trap Gates through conforming segments, or through segments with DPL > 0, will raise a GP fault with the CS selector as the error code.

4.6.6.1 Task Switches To/From Virtual 8086 Mode

Tasks which can execute in virtual 8086 mode must be described by a TSS with the new Intel486 Microprocessor format (TYPE 9 or 11 descriptor).

A task switch out of virtual 8086 mode will operate exactly the same as any other task switch out of a task with an Intel486 Microprocessor TSS. All of the programmer visible state, including the FLAGS register with the VM bit set to 1, is stored in the TSS. The segment registers in the TSS will contain 8086 segment base values rather than selectors.

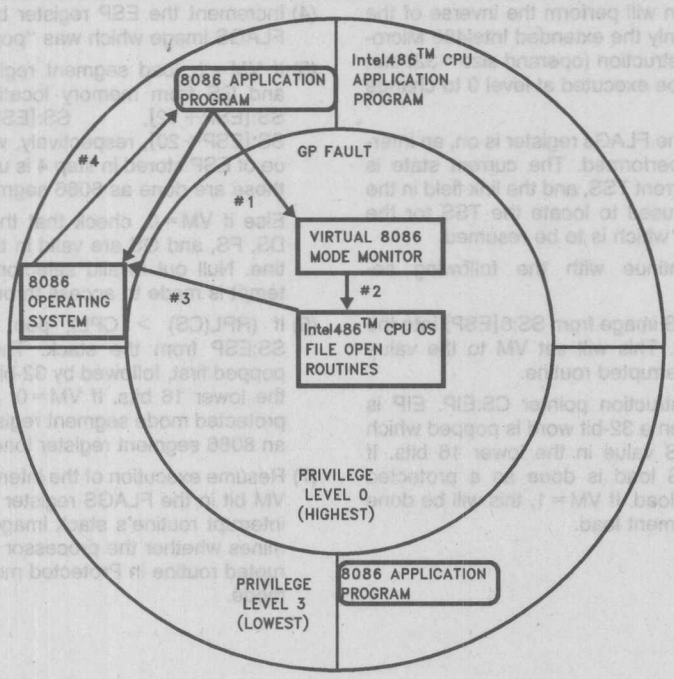
A task switch into a task described by an Intel486 Microprocessor TSS will have an additional check to determine if the incoming task should be resumed in virtual 8086 mode. Tasks described by 80286 format TSSs cannot be resumed in virtual 8086 mode, so no check is required there (the FLAGS image in 80286 format TSS has only the low order 16 FLAGS bits). Before loading the segment register images from an Intel486 Microprocessor TSS, the FLAGS image is loaded, so that the segment registers are loaded from the TSS image as 8086 segment base values. The task is now ready to resume in virtual 8086 execution mode.

4.6.6.2 Transitions Through Trap and Interrupt Gates, and IRET

A task switch is one way to enter or exit virtual 8086 mode. The other method is to exit through a Trap or Interrupt gate, as part of handling an interrupt, and to enter as part of executing an IRET instruction. The transition out must use an Intel486 Microprocessor Trap Gate (Type 14), or Intel486 Microprocessor Interrupt Gate (Type 15), which must point to a non-conforming level 0 segment (DPL = 0) in order to permit the trap handler to IRET back to the Virtual 8086 program. The Gate must point to a non-conforming level 0 segment to perform a level switch to level 0 so that the matching IRET can change the VM bit. Intel486 Microprocessor gates must be used, since 80286 gates save only the low 16 bits of the FLAGS register, so that the VM bit will not be saved on transitions through the 80286 gates. Also, the 16-bit IRET (presumably) used to terminate the 80286 interrupt handler will pop only the lower 16 bits from FLAGS, and will not affect the VM bit. The action taken for an Intel486 Microprocessor Trap or Interrupt gate if an interrupt occurs while the task is executing in virtual 8086 mode is given by the following sequence.

(1) Save the FLAGS register in a temp to push later. Turn off the VM and TF bits, and if the interrupt is serviced by an Interrupt Gate, turn off IF also.

(2) Interrupt and Trap gates must perform a level switch from 3 (where the VM86 program executes) to level 0 (so IRET can return). This process involves a stack switch to the stack given in the TSS for privilege level 0. Save the Virtual 8086 Mode SS and ESP registers to push in a later step. The segment register load of SS will be done as a Protected Mode segment load, since the VM bit was turned off above.



8086 Application makes "Open File Call" → causes
General Protection Fault (Arrow #1)
Virtual 8086 Monitor intercepts call. Calls Intel486™ CPU OS (Arrow #2)
Intel486™ CPU OS opens file returns control to 8086 OS (Arrow #3)
8086 OS returns control to application. (Arrow #4)
Transparent to Application

Figure 4.25. Virtual 8086 Environment Interrupt and Call Handling

(3) Push the 8086 segment register values onto the new stack, in the order: GS, FS, DS, ES. These are pushed as 32-bit quantities, with undefined values in the upper 16 bits. Then load these 4 registers with null selectors (0).

(4) Push the old 8086 stack pointer onto the new stack by pushing the SS register (as 32-bits, high bits undefined), then pushing the 32-bit ESP register saved above.

(5) Push the 32-bit FLAGS register saved in step 1.

(6) Push the old 8086 instruction pointer onto the new stack by pushing the CS register (as 32-bits, high bits undefined), then pushing the 32-bit EIP register.

(7) Load up the new CS:EIP value from the interrupt gate, and begin execution of the interrupt routine in protected Intel486 Microprocessor mode.

The transition out of virtual 8086 mode performs a level change and stack switch, in addition to changing back to protected mode. In addition, all of the 8086 segment register images are stored on the stack (behind the SS:ESP image), and then loaded with null (0) selectors before entering the interrupt handler. This will permit the handler to safely save and restore the DS, ES, FS, and GS registers as 80286 selectors. This is needed so that interrupt handlers which don't care about the mode of the interrupted program can use the same prolog and epilog code for state saving (i.e., push all registers in prolog, pop all in epilog) regardless of whether or not a "native" mode or Virtual 8086 mode program was interrupted. Restoring null selectors to these registers before executing the IRET will not cause a trap in the interrupt handler. Interrupt routines which expect values in the segment registers, or return values in segment registers will have to obtain/return values from the 8086 register images pushed onto the new stack. They will need to know the mode of the interrupted program in order to know where to find/return segment registers, and also to know how to interpret segment register values.

The IRET instruction will perform the inverse of the above sequence. Only the extended Intel486 Microprocessors IRET instruction (operand size = 32) can be used, and must be executed at level 0 to change the VM bit to 1.

(1) If the NT bit in the FLAGs register is on, an inter-task return is performed. The current state is stored in the current TSS, and the link field in the current TSS is used to locate the TSS for the interrupted task which is to be resumed.

Otherwise, continue with the following sequence.

(2) Read the FLAGS image from SS:8[ESP] into the FLAGS register. This will set VM to the value active in the interrupted routine.

(3) Pop off the instruction pointer CS:EIP. EIP is popped first, then a 32-bit word is popped which contains the CS value in the lower 16 bits. If VM = 0, this CS load is done as a protected mode segment load. If VM = 1, this will be done as an 8086 segment load.

(4) Increment the ESP register by 4 to bypass the FLAGS image which was "popped" in step 1.

(5) If VM = 1, load segment registers ES, DS, FS, and GS from memory locations SS:[ESP + 8], SS:[ESP + 12], SS:[ESP + 16], and SS:[ESP + 20], respectively, where the new value of ESP stored in step 4 is used. Since VM = 1, these are done as 8086 segment register loads.

Else if VM = 0, check that the selectors in ES, DS, FS, and GS are valid in the interrupted routine. Null out invalid selectors to trap if an attempt is made to access through them.

(6) If (RPL(CS) > CPL), pop the stack pointer SS:ESP from the stack. The ESP register is popped first, followed by 32-bits containing SS in the lower 16 bits. If VM = 0, SS is loaded as a protected mode segment register load. If VM = 1, an 8086 segment register load is used.

(7) Resume execution of the interrupted routine. The VM bit in the FLAGS register (restored from the interrupt routine's stack image in step 1) determines whether the processor resumes the interrupted routine in Protected mode of Virtual 8086 mode.

5.0 ON-CHIP CACHE

To meet its performance goals the Intel486 Microprocessor contains an eight Kbyte cache. The cache is software transparent to maintain binary compatibility with previous generations of the Intel386™/Intel486™ Architecture.

The on-chip cache has been designed for maximum flexibility and performance. The cache has several operating modes offering flexibility during program execution and debugging. Memory areas can be defined as non-cacheable by software and external hardware. Protocols for cache line invalidations and replacement are implemented in hardware, easing system design.

5.1 Cache Organization

The on-chip cache is a unified code and data cache. The cache is used for both instruction and data accesses and acts on physical addresses.

The cache organization is 4-way set associative and each line is 16 bytes wide. The eight Kbytes of cache memory are logically organized as 128 sets, each containing four lines.

The cache memory is physically split into four 2-Kbyte blocks each containing 128 lines (see Figure 5.1). Associated with each 2-Kbyte block are 128 21-bit tags. There is a valid bit for each line in the cache. Each line in the cache is either valid or not valid. There are no provisions for partially valid lines.

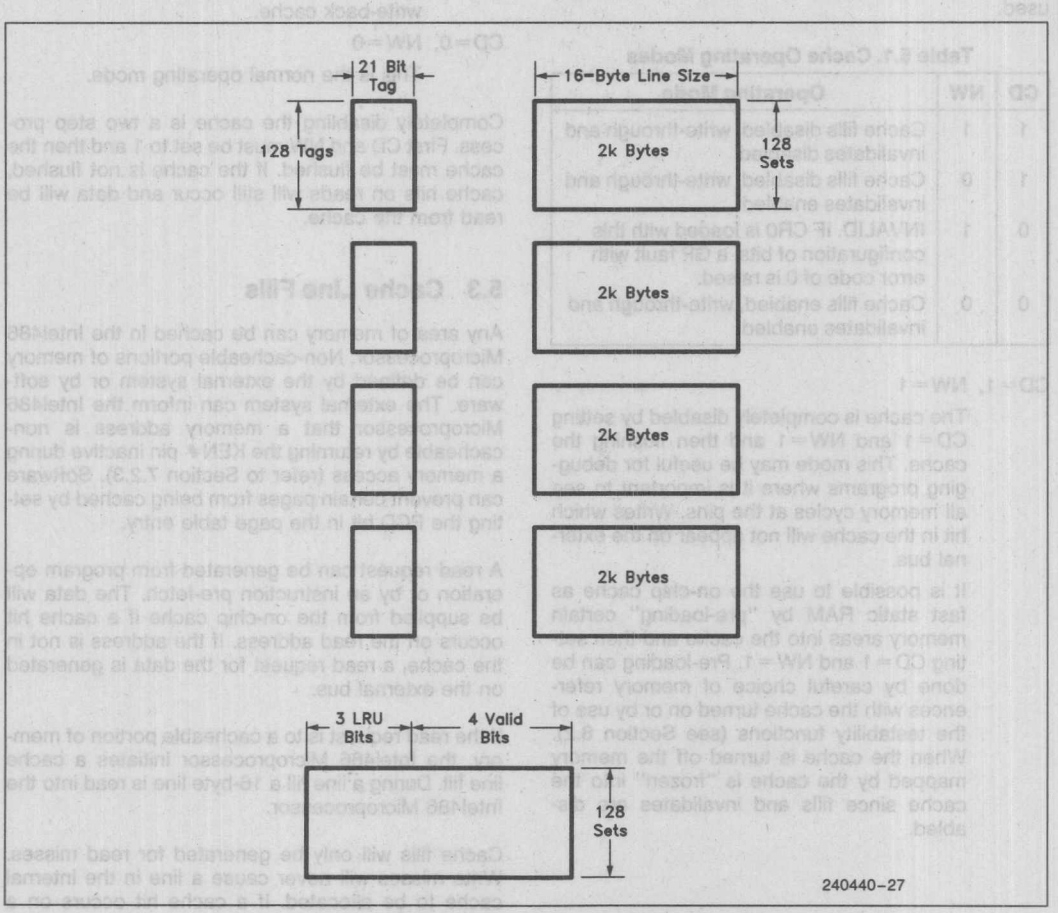


Figure 5.1. On-Chip Cache Physical Organization

The write strategy of on-chip cache is write-through. All writes will drive an external write bus cycle in addition to writing the information to the internal cache if the write was a cache hit. A write to an address not contained in the internal cache will only be written to external memory. Cache allocations are not made on write misses.

5.2 Cache Control

Control of the cache is provided by the CD and NW bits in CR0. CD enables and disables the cache. NW controls memory write-through and invalidates.

The CD and NW bits define four operating modes of the on-chip cache as given in Table 5.1. These modes provide flexibility in how the on-chip cache is used.

Table 5.1. Cache Operating Modes

CD	NW	Operating Mode
1	1	Cache fills disabled, write-through and invalidates disabled
1	0	Cache fills disabled, write-through and invalidates enabled
0	1	INVALID. IF CR0 is loaded with this configuration of bits, a GP fault with error code of 0 is raised.
0	0	Cache fills enabled, write-through and invalidates enabled

CD = 1, NW = 1

The cache is completely disabled by setting CD = 1 and NW = 1 and then flushing the cache. This mode may be useful for debugging programs where it is important to see all memory cycles at the pins. Writes which hit in the cache will not appear on the external bus.

It is possible to use the on-chip cache as fast static RAM by "pre-loading" certain memory areas into the cache and then setting CD = 1 and NW = 1. Pre-loading can be done by careful choice of memory references with the cache turned on or by use of the testability functions (see Section 8.2). When the cache is turned off the memory mapped by the cache is "frozen" into the cache since fills and invalidates are disabled.

CD = 1, NW = 0

Cache fills are disabled but write-throughs and invalidates are enabled. This mode is the same as if the KEN# pin was strapped HIGH disabling cache fills. Write-throughs and invalidates may still occur to keep the cache valid. This mode is useful if the software must disable the cache for a short period of time, and then re-enable it without flushing the original contents.

CD = 0, NW = 1

INVALID. If CR0 is loaded with this bit configuration, a General Protection fault with error code of 0 is raised. Note that this mode would imply a non-transparent write-back cache. A future processor may define this combination of bits to implement a write-back cache.

CD = 0, NW = 0

This is the normal operating mode.

Completely disabling the cache is a two step process. First CD and NW must be set to 1 and then the cache must be flushed. If the cache is not flushed, cache hits on reads will still occur and data will be read from the cache.

5.3 Cache Line Fills

Any area of memory can be cached in the Intel486 Microprocessor. Non-cacheable portions of memory can be defined by the external system or by software. The external system can inform the Intel486 Microprocessor that a memory address is non-cacheable by returning the KEN# pin inactive during a memory access (refer to Section 7.2.3). Software can prevent certain pages from being cached by setting the PCD bit in the page table entry.

A read request can be generated from program operation or by an instruction pre-fetch. The data will be supplied from the on-chip cache if a cache hit occurs on the read address. If the address is not in the cache, a read request for the data is generated on the external bus.

If the read request is to a cacheable portion of memory, the Intel486 Microprocessor initiates a cache line fill. During a line fill a 16-byte line is read into the Intel486 Microprocessor.

Cache fills will only be generated for read misses. Write misses will never cause a line in the internal cache to be allocated. If a cache hit occurs on a write, the line will be updated.

Cache line fills can be performed over 8- and 16-bit busses using the dynamic bus sizing feature. Refer to Section 7.1.3 for a description of dynamic bus sizing.

Refer to Section 7.2.3 for further information on cacheable cycles.

5.4 Cache Line Invalidations

The Intel486 Microprocessor contains both a hardware and software mechanism for invalidating lines in its internal cache. Cache line invalidations are needed to keep the Intel486 Microprocessor's cache contents consistent with external memory.

Refer to Section 7.2.8 for further information on cache line invalidations.

5.5 Cache Replacement

When a line needs to be placed in its internal cache the Intel486 Microprocessor first checks to see if there is a non-valid line in the set that can be replaced. If all four lines in the set are valid, a pseudo least-recently-used mechanism is used to determine which line should be replaced.

A valid bit is associated with each line in the cache. When a line needs to be placed in a set, the four valid bits are checked to see if there is a non-valid line that can be replaced. If a non-valid line is found, that line is marked for replacement.

The four lines in the set are labeled I0, I1, I2, and I3. The order in which the valid bits are checked during an invalidation is I0, I1, I2 and I3. All valid bits are cleared when the processor is reset or when the cache is flushed.

Replacement in the cache is handled by a pseudo least recently used (LRU) mechanism when all four lines in a set are valid. Three bits, B0, B1 and B2, are defined for each of the 128 sets in the cache. These bits are called the LRU bits. The LRU bits are updated for every hit or replace in the cache.

If the most recent access to the set was to I0 or I1, B0 is set to 1. B0 is set to 0 if the most recent access was to I2 or I3. If the most recent access to I0:I1 was to I0, B1 is set to 1, else B1 is set to 0. If the most recent access to I2:I3 was to I2, B2 is set to 1, else B2 is set to 0.

The pseudo LRU mechanism works in the following manner. When a line must be replaced, the cache will first select which of I0:I1 and I2:I3 was least recently used. Then the cache will determine which of the two lines was least recently used and mark it for replacement. This decision tree is shown in Figure 5.2. When the processor is reset or when the cache is flushed all 128 sets of three LRU bits are set to 0.

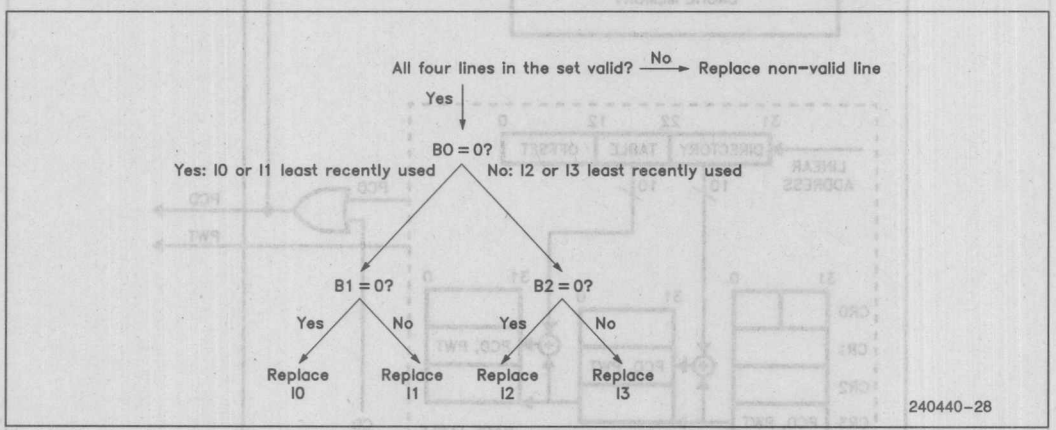


Figure 5.2. On-Chip Cache Replacement Strategy

5.6 Page Cacheability

Two bits for cache control, PWT and PCD, are defined in the page table and page directory entries. The state of these bits are driven out on the PWT and PCD pins during memory access cycles.

The PWT bit controls write policy for second level caches used with the Intel486 Microprocessor. Setting PWT = 1 defines a write-through policy for the current page while PWT = 0 allows the possibility of write-back. The state of PWT is ignored internally by the Intel486 Microprocessor since the on-chip cache is write through.

The PCD bit controls cacheability on a page by page basis. The PCD bit is internally ANDed with the KEN# signal to control cacheability on a cycle by cycle basis (see Figure 5.3). PCD = 0 enables caching while PCD = 1 forbids it. Note that cache fills are enabled when PCD = 0 AND KEN# = 0. This logical AND is implemented physically with a NOR gate.

The state of the PCD bit in the page table entry is driven on the PCD pin when a page in external memory is accessed. The state of the PCD pin informs the external system of the cacheability of the requested information. The external system then returns KEN# telling the Intel486 Microprocessor if the area is cacheable. The Intel486 Microprocessor initiates a cache line fill if PCD and KEN# indicate that the requested information is cacheable.

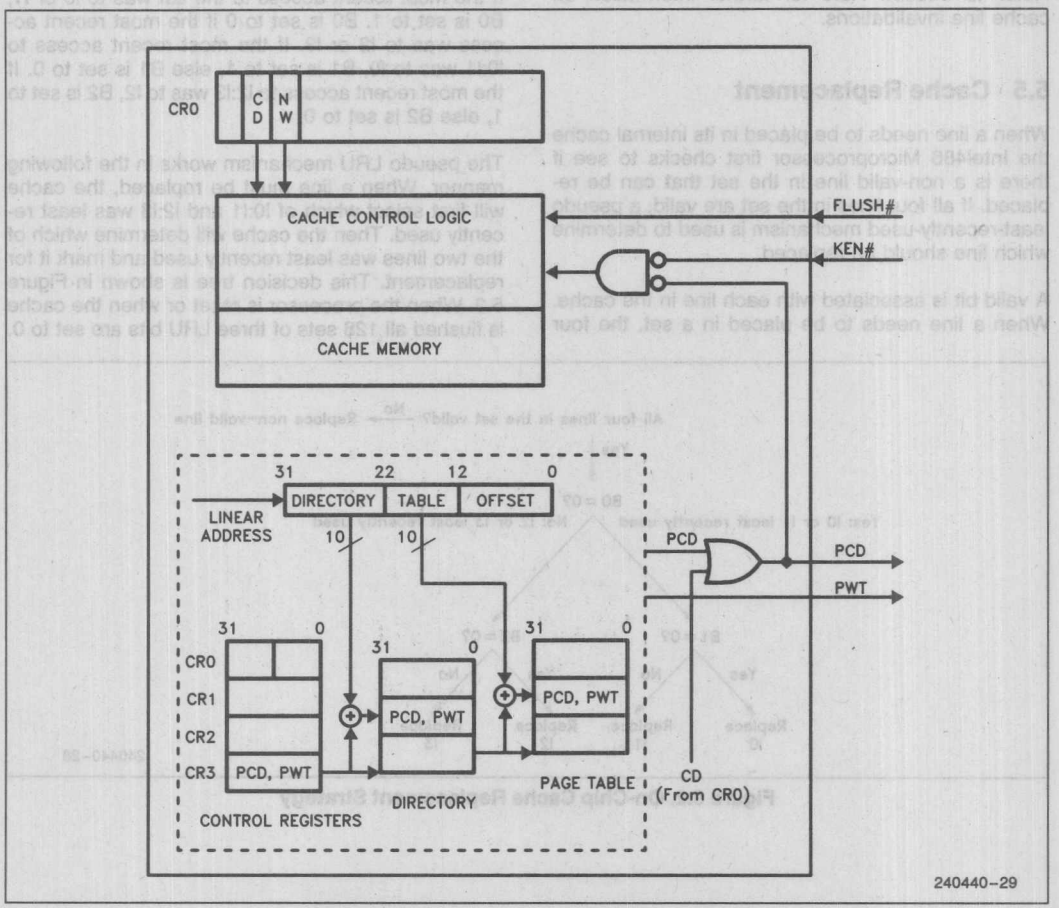


Figure 5.3. Page Cacheability

The PCD bit is masked with the CD (cache disable) bit in control register 0 to determine the state of the PCD pin. If CD=1 the Intel486 Microprocessor forces the PCD pin HIGH. If CD=0 the PCD pin is driven with the value for the page table entry/directory. See Figure 5.3.

The PWT and PCD bits for a bus cycle are obtained from either CR3, the page directory or page table entry. These bits are assumed to be zero during real mode, whenever paging is disabled, or for cycles that bypass paging, (I/O references, interrupt acknowledge and Halt cycles), the PWT and PCD bits are taken from CR3. These bits are initialized to 0 on reset, but can be set to any value by level 0 software.

When paging is enabled, the bits from the page table entry are cached in the TLB, and are driven any time the page mapped by the TLB entry is referenced. For normal memory cycles, PWT and PCD are taken from the page table entry. During TLB refresh cycles where the page table and directory entries are read, the PWT and PCD bits must be obtained elsewhere. During page table updates the bits are obtained from the page directory. When the page directory is updated the bits are obtained from CR3.

5.7 Cache Flushing

The on-chip cache can be flushed by external hardware or by software instructions. Flushing the cache clears all valid bits for all lines in the cache. The cache is flushed when external hardware asserts the FLUSH# pin.

The flush pin needs to be asserted for one clock if driven synchronously or for two clocks if driven asynchronously. The flush input is asynchronous but setup and hold times must be met. The flush pin should be deasserted after the cache flush is complete. Failure to deassert the pin will cause execution to stop as the processor will be repeatedly flushing the cache. If external hardware activates flush in response to an I/O write, flush must be asserted for at least two clocks prior to ready being returned for the I/O write. This ensures that the flush completes before the CPU begins execution of the instruction following the OUT instruction.

Flush is recognized during HOLD just like EADS#.

The instructions INVD and WBINVD cause the on-cache to be flushed. External caches connected to the Intel486 microprocessor are signalled to flush their contents when these instructions are executed.

WBINVD will cause an external write-back cache to write back dirty lines before flushing its contents. The external cache is signalled using the bus cycle definition pins and the byte enables (refer to Section 6.2.5 for the bus cycle definition pins and Section 7.2.11 for special bus cycles). Refer to the Intel486 Microprocessor programmers reference manual for detailed instruction definitions.

The results of the INVD and WBINVD instructions are identical for the operation of the Intel486 Microprocessor's on-chip cache since the cache is write-through. Note that the INVD and WBINVD instructions are machine dependent. Future members of the Intel486 Microprocessor family may change the definition of this instruction.

5.8 Caching Translation Lookaside Buffer Entries

The Intel486 Microprocessor contains an integrated paging unit with a translation lookaside buffer (TLB). The TLB contains 32 entries. The TLB has been enhanced over the 386 Microprocessor's TLB by upgrading the replacement strategy to a pseudo-LRU (least recently used) algorithm. The pseudo-LRU replacement algorithm is the same as that used in the on-chip cache.

The paging TLB operation is automatic whenever paging is enabled. The TLB contains the most recently used page table entries. A page table entry translates the linear address pointing to a particular page to the physical address where the page is stored in memory (refer to Section 4.5, **Paging**).

The paging unit will look up the linear address in the TLB in response to an internal bus request. The corresponding physical address is passed on to the on-chip cache or the external bus (in the event of a cache miss) when the linear address is present in the TLB.

The paging unit will access the page tables in external memory if the linear address is not in the TLB. The required page table entry will be read into the TLB and then the cache or bus cycle for the actual data will take place. The process of reading a new page table entry into the TLB is called a TLB refresh.

A TLB refresh is a two step process. The paging unit must first read the page directory entry which points to the appropriate page table. The page table entry to be stored in the TLB is then read from the page table. Control register 3 (CR3) points to the base of the page directory table.

The Intel486 Microprocessor will allow page directory and page table entries (returned during TLB refreshes) to be stored in the on-chip cache. Setting the PCD bits in CR3 and the page directory entry to 1 will prevent the page directory and page table entries from being stored in the on-chip cache (see Section 5.6, **Page Cacheability**).

6.0 HARDWARE INTERFACE

6.1 Introduction

The Intel486 Microprocessor bus has been designed to be similar to the 386 Microprocessor bus whenever possible. Several new features have been added to the Intel486 Microprocessor bus resulting in increased performance and functionality. New features include a 1X clock, a burst bus mechanism for high-speed internal cache fills, a cache line invalidation mechanism, enhanced bus arbitration capabilities, a BS8# bus sizing mechanism and parity support.

The Intel486 Microprocessor is driven by a 1X clock as opposed to a 2X clock in the 386 Microprocessor. A 25 MHz Intel486 Microprocessor uses a 25 MHz clock in contrast to a 25 MHz 386 Microprocessor which requires a 50 MHz clock. A 1X clock allows simpler system design by cutting in half the clock speed required in the external system.

Like the 386 Microprocessor, the Intel486 Microprocessor has separate parallel busses for data and addresses. The bidirectonal data bus is 32 bits in width. The address bus consists of two components: 30 address lines (A2–A31) and 4 byte enable lines (BE0#–BE3#). The address bus addresses external memory in the same manner as the 386 Microprocessor: The address lines form the upper 30 bits of the address and the byte enables select individual bytes within a 4 byte location. The address lines are bidirectional for use in cache line invalidations.

The Intel486 Microprocessor's burst bus mechanism enables high-speed cache fills from external memory. Burst cycles can strobe data into the processor at a rate of one item every clock. Non-burst cycles have a maximum rate of one item every two clocks. Burst cycles are not limited to cache fills: all bus cycles requiring more than a single data cycle can be bursted.

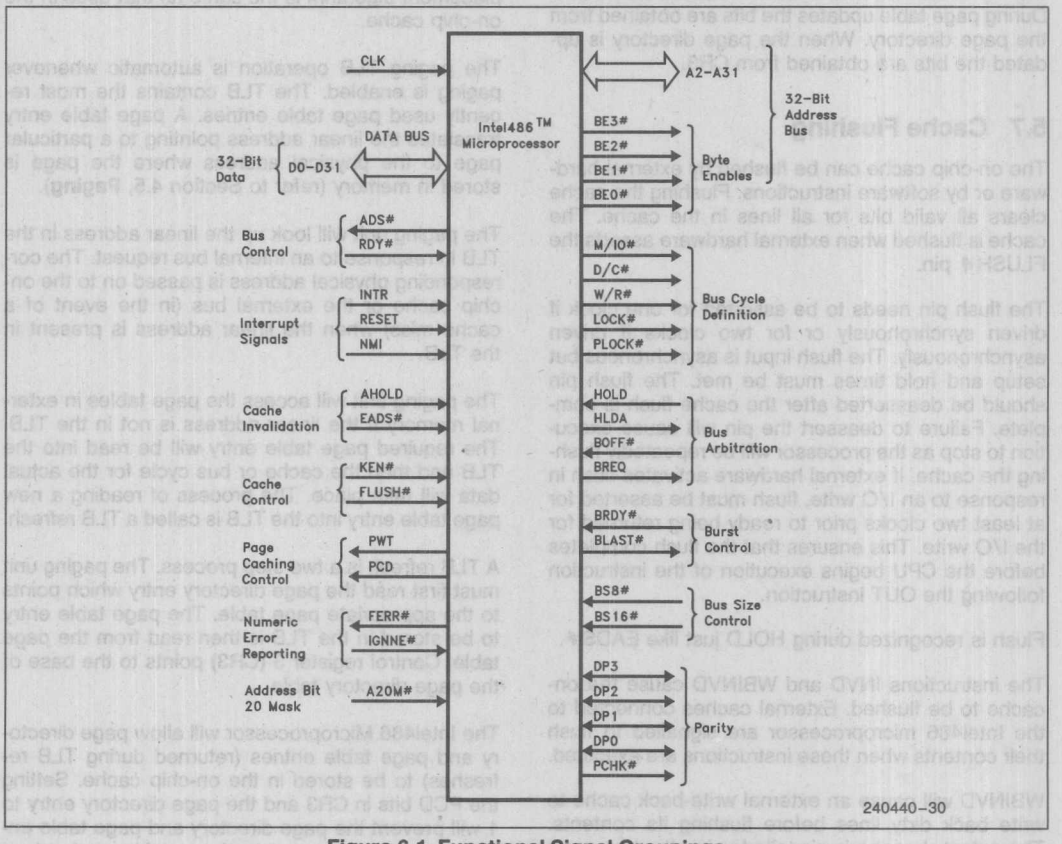


Figure 6.1. Functional Signal Groupings

The Intel486 Microprocessor has a bus hold feature similar to that of the 386 Microprocessor. During bus hold, the Intel486 Microprocessor relinquishes control of the local bus by floating its address, data and control busses.

The Intel486 Microprocessor has an address hold feature in addition to bus hold. During address hold only the address bus is floated, the data and control busses can remain active. Address hold is used for cache line invalidations.

Ahead is a brief description of the Intel486 Microprocessor input and output signals arranged by functional groups. Before beginning the signal descriptions a few terms need to be defined. The # symbol at the end of a signal name indicates the active, or asserted, state occurs when the signal is at a low voltage. When a # is not present after the signal name, the signal is active at the high voltage level. The term "ready" is used to indicate that the cycle is terminated with RDY# or BRDY#.

Section 6 and 7 will discuss bus cycles and data cycles. A bus cycle is at least two clocks long and begins with ADS# active in the first clock and ready active in the last clock. Data is transferred to or from the Intel486 Microprocessor during a data cycle. A bus cycle contains one or more data cycles.

6.2 Signal Descriptions

6.2.1 CLOCK (CLK)

CLK provides the fundamental timing and the internal operating frequency for the Intel486 Microprocessor. All external timing parameters are specified with respect to the rising edge of CLK.

The Intel486 Microprocessor can operate over a wide frequency range but CLK's frequency cannot change rapidly while RESET is inactive. CLK's frequency must be stable for proper chip operation since a single edge of CLK is used internally to generate two phases. CLK only needs TTL levels for proper operation. Figure 6.2 illustrates the CLK waveform.

6.2.2 ADDRESS BUS (A31–A2, BE0#–BE3#)

A31–A2 and BE0#–BE3# form the address bus and provide physical memory and I/O port addresses. The Intel486 Microprocessor is capable of addressing 4 gigabytes of physical memory space (00000000H through FFFFFFFFH), and 64 Kbytes of I/O address space (00000000H through 0000FFFFH). A31–A2 identify addresses to a 4-byte location. BE0#–BE3# identify which bytes within the 4-byte location are involved in the current transfer.

Addresses are driven back into the Intel486 Microprocessor over A31–A4 during cache line invalidations. The address lines are active HIGH. When used as inputs into the processor, A31–A4 must meet the setup and hold times, t_{22} and t_{23}. A31–A2 are not driven during bus or address hold.

The byte enable outputs, BE0#–BE3#, determine which bytes must be driven valid for read and write cycles to external memory.

BE3# applies to D24–D31
BE2# applies to D16–D23
BE1# applies to D8–D15
BE0# applies to D0–D7

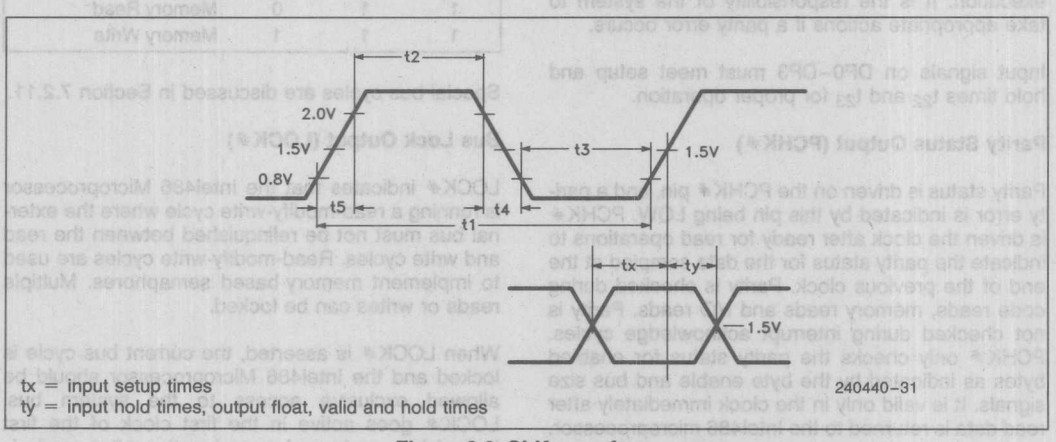


Figure 6.2. CLK waveform

BE0#–BE3# can be decoded to generate A0, A1 and BHE# signals used in 8- and 16-bit systems (see Table 7.5). BE0#–BE3# are active LOW and are not driven during bus hold.

6.2.3 DATA LINES (D31–D0)

The bidirectional lines, D31–D0, form the data bus for the Intel486 Microprocessor. D0–D7 define the least significant byte and D24–D31 the most significant byte. Data transfers to 8- or 16-bit devices is possible using the data bus sizing feature controlled by the BS8# or BS16# input pins.

D31–D0 are active HIGH. For reads, D31–D0 must meet the setup and hold times, t_{22} and t_{23}. D31–D0 are not driven during read cycles and bus hold.

6.2.4 PARITY

Data Parity Input/Outputs (DP0–DP3)

DP0–DP3 are the data parity pins for the processor. There is one pin for each byte of the data bus. Even parity is generated or checked by the parity generators/checkers. Even parity means that there are an even number of HIGH inputs on the eight corresponding data bus pins and parity pin.

Data parity is generated on all write data cycles with the same timing as the data driven by the Intel486 Microprocessor. Even parity information must be driven back to the Intel486 Microprocessor on these pins with the same timing as read information to insure that the correct parity check status is indicated by the Intel486 Microprocessor.

The values read on these pins do not affect program execution. It is the responsibility of the system to take appropriate actions if a parity error occurs.

Input signals on DP0–DP3 must meet setup and hold times t_{22} and t_{23} for proper operation.

Parity Status Output (PCHK#)

Parity status is driven on the PCHK# pin, and a parity error is indicated by this pin being LOW. PCHK# is driven the clock after ready for read operations to indicate the parity status for the data sampled at the end of the previous clock. Parity is checked during code reads, memory reads and I/O reads. Parity is not checked during interrupt acknowledge cycles. PCHK# only checks the parity status for enabled bytes as indicated by the byte enable and bus size signals. It is valid only in the clock immediately after read data is returned to the Intel486 microprocessor. At all other times it is inactive (HIGH). PCHK# is never floated.

Driving PCHK# is the only effect that bad input parity has on the Intel486 Microprocessor. The Intel486 Microprocessor will not vector to a bus error interrupt when bad data parity is returned. In systems that will not employ parity, PCHK# can be ignored. In systems not using parity, DP0–DP3 should be connected to V_{CC} through a pullup resistor.

6.2.5 BUS CYCLE DEFINITION

M/IO#, D/C#, W/R# Outputs

M/IO#, D/C# and W/R# are the primary bus cycle definition signals. They are driven valid as the ADS# signal is asserted. M/IO# distinguishes between memory and I/O cycles, D/C# distinguishes between data and control cycles and W/R# distinguishes between write and read cycles.

Bus cycle definitions as a function of M/IO#, D/C# and W/R# are given in Table 6.1. Note there is a difference between the Intel486 Microprocessor and 386 Microprocessor bus cycle definitions. The halt bus cycle type has been moved to location 001 in the Intel486 Microprocessor from location 101 in the 386 Microprocessor. Location 101 is now reserved and will never be generated by the Intel486 Microprocessor.

Table 6.1. ADS# Initiated Bus Cycle Definitions

M/IO#	D/C#	W/R#	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Halt/Special Cycle
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Code Read
1	0	1	Reserved
1	1	0	Memory Read
1	1	1	Memory Write

Special bus cycles are discussed in Section 7.2.11.

Bus Lock Output (LOCK#)

LOCK# indicates that the Intel486 Microprocessor is running a read-modify-write cycle where the external bus must not be relinquished between the read and write cycles. Read-modify-write cycles are used to implement memory-based semaphores. Multiple reads or writes can be locked.

When LOCK# is asserted, the current bus cycle is locked and the Intel486 Microprocessor should be allowed exclusive access to the system bus. LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after ready is returned indicating the last locked bus cycle.

The Intel486 Microprocessor will not acknowledge bus hold when LOCK# is asserted (though it will allow an address hold). LOCK# is active LOW and is floated during bus hold. Locked read cycles will not be transformed into cache fill cycles if KEN# is returned active. Refer to Section 7.2.6 for a detailed discussion of Locked bus cycles.

Pseudo-Lock Output (PLOCK#)

The pseudo-lock feature allows atomic reads and writes of memory operands greater than 32 bits. These operands require more than one cycle to transfer. The Intel486 Microprocessor asserts PLOCK# during floating point long reads and writes (64 bits), segment table descriptor reads (64 bits) and cache line fills (128 bits).

When PLOCK# is asserted no other master will be given control of the bus between cycles. A bus hold request (HOLD) is not acknowledged during pseudo-locked reads and writes, with one exception. During non-cacheable non-bursted code prefetches, HOLD is recognized on memory cycle boundaries even though PLOCK# is asserted. The Intel486 Microprocessor will drive PLOCK# active until the addresses for the last bus cycle of the transaction have been driven regardless of whether BRDY# or RDY# are returned.

A pseudo-locked transfer is meaningful only if the memory operand is aligned and if its completely contained within a single cache line. A 64-bit floating point number must be aligned to an 8-byte boundary to guarantee an atomic access.

Normally PLOCK# and BLAST# are inverse of each other. However during the first cycle of a 64-bit floating point write, both PLOCK# and BLAST# will be asserted.

Since PLOCK# is a function of the bus size and KEN# inputs, PLOCK# should be sampled only in the clock ready is returned. This pin is active LOW and is not driven during bus hold. Refer to Section 7.2.7 for a detailed discussion of pseudo-locked bus cycles.

6.2.6 BUS CONTROL

The bus control signals allow the processor to indicate when a bus cycle has begun, and allow other system hardware to control burst cycles, data bus width and bus cycle termination.

Address Status Output (ADS#)

The ADS# output indicates that the address and bus cycle definition signals are valid. This signal will go active in the first clock of a bus cycle and go inactive in the second and subsequent clocks of the cycle. ADS# is also inactive when the bus is idle.

ADS# is used by external bus circuitry as the indication that the processor has started a bus cycle. The external circuit must sample the bus cycle definition pins on the next rising edge of the clock after ADS# is driven active.

ADS# is active LOW and is not driven during bus hold.

Non-burst Ready Input (RDY#)

RDY# indicates that the current bus cycle is complete. In response to a read, RDY# indicates that the external system has presented valid data on the data pins. In response to a write request, RDY# indicates that the external system has accepted the Intel486 microprocessor data. RDY# is ignored when the bus is idle and at the end of the first clock of the bus cycle. Since RDY# is sampled during address hold, data can be returned to the processor when AHOLD is active.

RDY# is active LOW, and is not provided with an internal pullup resistor. This input must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.

6.2.7 BURST CONTROL

Burst Ready Input (BRDY#)

BRDY# performs the same function during a burst cycle that RDY# performs during a non-burst cycle. BRDY# indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted the Intel486 Microprocessor data in response to a write. BRDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle.

During a burst cycle, BRDY# will be sampled each clock, and if active, the data presented on the data bus pins will be strobed into the Intel486 Microprocessor. ADS# is negated during the second through last data cycles in the burst, but address lines A2–A3 and byte enables will change to reflect the next data item expected by the Intel486 Microprocessor.

If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is prematurely aborted. An additional complete bus cycle will be initiated after an aborted burst cycle if the cache line fill was not complete. BRDY# is treated as a normal ready for the last data cycle in a burst transfer or for non-burstable cycles. Refer to Section 7.2.2 for burst cycle timing.

BRDY# is active LOW and is provided with a small internal pullup resistor. BRDY# must satisfy the set-up and hold times t_{16} and t_{17}.

Burst Last Output (BLAST#)

BLAST# indicates that the next time BRDY# is returned it will be treated as a normal RDY#, terminating the line fill or other multiple-data-cycle transfer. BLAST# is active for all bus cycles regardless of whether they are cacheable or not. This pin is active LOW and is not driven during bus hold.

6.2.8 INTERRUPT SIGNALS (RESET, INTR, NMI)

The interrupt signals can interrupt or suspend execution of the processor's current instruction stream.

Reset Input (RESET)

RESET forces the Intel486 Microprocessor to begin execution at a known state. For a power-up (cold start) reset, V_{CC} and CLK must reach their proper DC and AC specifications for at least 1 ms before the Intel486 Microprocessor begins instruction execution. The RESET pin should remain active during this time to ensure proper Intel486 Microprocessor operation. However, for a warm boot-up case, RESET is required to remain active for a minimum of 15 clocks. The testability operating modes are programmed by the falling (inactive going) edge of RESET. (Refer to Section 8.0 for a description of the test modes during reset.)

Maskable Interrupt Request Input (INTR)

INTR indicates that an external interrupt has been generated. Interrupt processing is initiated if the IF flag is active in the EFLAGS register.

The Intel486 Microprocessor will generate two locked interrupt acknowledge bus cycles in response to asserting the INTR pin. An 8-bit interrupt number will be latched from an external interrupt controller at the end of the second interrupt acknowledge cycle. INTR must remain active until the interrupt acknowledges have been performed to assure program interruption. Refer to Section 7.2.10 for a detailed discussion of interrupt acknowledge cycles.

The INTR pin is active HIGH and is not provided with an internal pulldown resistor. INTR is asynchronous, but the INTR setup and hold times, t_{20} and t_{21}, must be met to assure recognition on any specific clock.

Non-maskable Interrupt Request Input (NMI)

NMI is the non-maskable interrupt request signal. Asserting NMI causes an interrupt with an internally supplied vector value of 2. External interrupt acknowledge cycles are not generated since the NMI interrupt vector is internally generated. When NMI processing begins, the NMI signal will be masked internally until the IRET instruction is executed.

NMI is rising edge sensitive after internal synchronization. NMI must be held LOW for at least four CLK periods before this rising edge for proper operation. NMI is not provided with an internal pulldown resistor. NMI is asynchronous but setup and hold times, t_{20} and t_{21} must be met to assure recognition on any specific clock.

6.2.9 BUS ARBITRATION SIGNALS

This section describes the mechanism by which the processor relinquishes control of its local bus when requested by another bus master.

Bus Request Output (BREQ)

The Intel486 Microprocessor asserts BREQ whenever a bus cycle is pending internally. Thus, BREQ is always asserted in the first clock of a bus cycle, along with ADS#. Furthermore, if the Intel486 Microprocessor is currently not driving the bus (due to HOLD, AHOLD, or BOFF#), BREQ is asserted in the same clock that ADS# would have been asserted if the processor were driving the bus. After the first clock of the bus cycle, BREQ may change state. It will be asserted if additional cycles are necessary to complete a transfer (via BS8#, BS16#, KEN#), or if more cycles are pending internally. However, if no additional cycles are necessary to complete the current transfer, BREQ can be negated before ready comes back for the current cycle. External logic can use the BREQ signal to arbitrate among multiple processors. This pin is driven regardless of the state of bus hold or address hold. BREQ is active HIGH and is never floated. During a hold state, internal events may cause BREQ to be deasserted prior to any bus cycles.

Bus Hold Request Input (HOLD)

HOLD allows another bus master complete control of the Intel486 Microprocessor bus. The Intel486 Microprocessor will respond to an active HOLD signal by asserting HLDA and placing most of its output and input/output pins in a high impedance state (floated) after completing its current bus cycle, burst cycle, or sequence of locked cycles. In addition, if the Intel486 CPU receives a HOLD request while performing a code fetch, and that cycle is backed off (BOFF#), the Intel486 CPU will recognize HOLD before restarting the cycle. The BREQ, HLDA, PCHK# and FERR# pins are not floated during bus hold. The Intel486 Microprocessor will maintain its bus in this state until the HOLD is deasserted. Refer to

Section 7.2.9 for timing diagrams for bus hold cycles and HOLD request acknowledge during BOFF#.

Unlike the 386 Microprocessor, the Intel486 Microprocessor will recognize HOLD during reset. Pullup resistors are not provided for the outputs that are floated in response to HOLD. HOLD is active HIGH and is not provided with an internal pulldown resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper chip operation.

Bus Hold Acknowledge Output (HLDA)

HLDA indicates that the Intel486 Microprocessor has given the bus to another local bus master. HLDA goes active in response to a hold request presented on the HOLD pin. HLDA is driven active in the same clock that the Intel486 Microprocessor floats its bus.

HLDA will be driven inactive when leaving bus hold and the Intel486 Microprocessor will resume driving the bus. The Intel486 Microprocessor will not cease internal activity during bus hold since the internal cache will satisfy the majority of bus requests. HLDA is active HIGH and remains driven during bus hold.

Backoff Input (BOFF#)

Asserting the BOFF# input forces the Intel486 Microprocessor to release control of its bus in the next clock. The pins floated are exactly the same as in response to HOLD. The response to BOFF# differs from the response to HOLD in two ways: First, the bus is floated immediately in response to BOFF# while the Intel486 Microprocessor completes the current bus cycle before floating its bus in response to HOLD. Second the Intel486 does not assert HLDA in response to BOFF#.

The processor remains in bus hold until BOFF# is negated. Upon negation, the Intel486 Microprocessor restarts the bus cycle aborted when BOFF# was asserted. To the internal execution engine the effect of BOFF# is the same as inserting a few wait states to the original cycle. Refer to Section 7.2.12 for a description of bus cycle restart.

Any data returned to the processor while BOFF# is asserted is ignored. BOFF# has higher priority than RDY# or BRDY#. If both BOFF# and ready are returned in the same clock, BOFF# takes effect. If BOFF# is asserted while the bus is idle, the Intel486 Microprocessor will float its bus in the next clock. BOFF# is active LOW and must meet setup and hold times t_{18} and t_{19} for proper chip operation.

6.2.10 CACHE INVALIDATION

The AHOLD and EADS# inputs are used during cache invalidation cycles. AHOLD conditions the Intel486 Microprocessors address lines, A4–A31, to accept an address input. EADS# indicates that an external address is actually valid on the address inputs. Activating EADS# will cause the Intel486 Microprocessor to read the external address bus and perform an internal cache invalidation cycle to the address indicated. Refer to Section 7.2.8 for cache invalidation cycle timing.

Address Hold Request Input (AHOLD)

AHOLD is the address hold request. It allows another bus master access to the Intel486 Microprocessor address bus for performing an internal cache invalidation cycle. Asserting AHOLD will force the Intel486 Microprocessor to stop driving its address bus in the next clock. While AHOLD is active only the address bus will be floated, the remainder of the bus can remain active. For example, data can be returned for a previously specified bus cycle when AHOLD is active. The Intel486 Microprocessor will not initiate another bus cycle during address hold. Since the Intel486 Microprocessor floats its bus immediately in response to AHOLD, an address hold acknowledge is not required. If AHOLD is asserted while a bus cycle is in progress, and no readies are returned during the time AHOLD is asserted, the Intel486 will redrive the same address (that it originally sent out) once AHOLD is negated.

AHOLD is recognized during reset. Since the entire cache is invalidated by reset, any invalidation cycles run during reset will be unnecessary. AHOLD is active HIGH and is provided with a small internal pulldown resistor. It must satisfy the setup and hold times t_{18} and t_{19} for proper chip operation. This pin determines whether or not the built in self test features of the Intel486 Microprocessor will be exercised on assertion of RESET.

External Address Valid Input (EADS#)

EADS# indicates that a valid external address has been driven onto the Intel486 address pins. This address will be used to perform an internal cache invalidation cycle. The external address will be checked with the current cache contents. If the address specified matches any areas in the cache, that area will immediately be invalidated.

An invalidation cycle may be run by asserting EADS# regardless of the state of AHOLD, HOLD and BOFF#. EADS# is active LOW and is provided with an internal pullup resistor. EADS# must satisfy the setup and hold times t_{12} and t_{13} for proper chip operation.

6.2.11 CACHE CONTROL

Cache Enable Input (KEN#)

KEN# is the cache enable pin. KEN# is used to determine whether the data being returned by the current cycle is cacheable. When KEN# is active and the Intel486 Microprocessor generates a cycle that can be cached (most any memory read cycle), the cycle will be transformed into a cache line fill cycle.

A cache line is 16 bytes long. During the first cycle of a cache line fill the byte-enable pins should be ignored and data should be returned as if all four byte enables were asserted. The Intel486 Microprocessor will run between 4 and 16 contiguous bus cycles to fill the line depending on the bus data width selected by BS8# and BS16#. Refer to Section 7.2.3 for a description of cache line fill cycles.

The KEN# input is active LOW and is provided with a small internal pullup resistor. It must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

Cache Flush Input (FLUSH#)

The FLUSH# input forces the Intel486 Microprocessor to flush its entire internal cache. FLUSH# is active LOW and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition on any specific clock.

FLUSH# also determines whether or not the tristate test mode of the Intel486 Microprocessor will be invoked on assertion of RESET.

6.2.12 PAGE CACHEABILITY (PWT, PCD)

The PWT and PCD output signals correspond to two user attribute bits in the page table entry. When paging is enabled, PWT and PCD correspond to bits 3 and 4 of the page table entry respectively. For cycles that are not paged when paging is enabled (for example I/O cycles) PWT and PCD correspond to bits 3 and 4 in control register 3. When paging is disabled, the Intel486 CPU ignores the PCD and PWT bits and assumes they are zero for the purpose of caching and driving PCD and PWT.

PCD is masked by the CD (cache disable) bit in control register 0 (CR0). When CD = 1 (cache line fills disabled) the Intel486 Microprocessor forces PCD HIGH. When CD = 0, PCD is driven with the value of the page table entry/directory.

The purpose of PCD is to provide a cacheable/non-cacheable indication on a page by page basis. The Intel486 will not perform a cache fill to any page in which bit 4 of the page table entry is set. PWT corresponds to the write-back bit and can be used by an external cache to provide this functionality. PCD and PWT bits are assigned to be zero during real mode or whenever paging is disabled. Refer to Sections 4.5.4 and 5.6 for a discussion of non-cacheable pages.

PCD and PWT have the same timing as the cycle definition pins (M/IO#, D/C#, W/R#). PCD and PWT are active HIGH and are not driven during bus hold.

6.2.13 NUMERIC ERROR REPORTING (FERR#, IGNNE#)

To allow PC-type floating point error reporting, the Intel486 Microprocessor provides two pins, FERR# and IGNNE#.

Floating Point Error Output (FERR#)

The Intel486 Microprocessor asserts FERR# whenever an unmasked floating point error is encountered. FERR# is similar to the ERROR# pin on the 387 Math Coprocessor. FERR# can be used by external logic for PC-type floating point error reporting in Intel486 Microprocessor systems. FERR# is active LOW, and is not floated during bus hold.

In some cases, FERR# is asserted when the next floating point instruction is encountered and in other cases it is asserted before the next floating point instruction is encountered depending upon the execution state of the instruction causing the exception.

The following class of floating point exceptions drive FERR# at the time the exception occurs (i.e., before encountering the next floating point instruction).

1. The stack fault, invalid operation, and denormal exceptions on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exceptions on store instructions (including integer store instructions).

The following class of floating point exceptions drive FERR# only after encountering the next floating point instruction.

1. Exceptions other than on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exception on all basic arithmetic, load, compare, and control instructions (i.e., all other instructions).

Ignore Numeric Error Input (IGNNE#)

The Intel486 Microprocessor will ignore a numeric error and continue executing non-control floating point instructions when IGNNE# is asserted, but FERR# will still be activated. When deasserted, the Intel486 Microprocessor will freeze on a non-control floating point instruction if a previous instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set.

The IGNNE# input is active LOW and is provided with a small internal pullup resistor. This input is asynchronous, but must meet setup and hold times t_{20} and t_{21} to insure recognition on any specific clock.

6.2.14 BUS SIZE CONTROL (BS16#, BS8#)

The BS16# and BS8# inputs allow external 16- and 8-bit busses to be supported with a small number of external components. The Intel486 CPU samples these pins every clock. The value sampled in the clock before ready determines the bus size. When asserting BS16# or BS8# only 16 or 8 bits of the data bus need be valid. If both BS16# and BS8# are asserted, an 8-bit bus width is selected.

When BS16# or BS8# are asserted the Intel486 Microprocessor will convert a larger data request to the appropriate number of smaller transfers. The byte enables will also be modified appropriately for the bus size selected.

BS16# and BS8# are active LOW and are provided with small internal pullup resistors. BS16# and BS8# must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

6.2.15 ADDRESS BIT 20 MASK (A20M#)

Asserting the A20M# input causes the Intel486 Microprocessor to mask physical address bit 20 before performing a lookup in the internal cache and before driving a memory cycle to the outside world. When A20M# is asserted, the Intel486 Microprocessor emulates the 1 Mbyte address wraparound that occurs on the 8086. A20M# is active LOW and must be asserted only when the processor is in real mode. The A20M# is not defined in Protected Mode. A20M# is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For correct operation of the chip, A20M# should be sampled high 2 clocks before and 2 clocks after RESET goes low. When A20M# is asserted synchronously, A20M# should be high (non-active) at the clock prior to the falling edge of RESET. A20M# exhibits a minimum 4 clock latency, from time of assertion to masking of the A20 bit. A20M# is ignored during cache invalidation cycles. I/O writes require A20M# to be asserted a minimum of 2 clocks prior to RDY being returned for the I/O write. This insures recognition of the address mask before the i486 SX Microprocessor/Intel OverDrive Processor begins execution of the instruction following OUT. If A20M# is asserted after the ADS# of a data cycle, the A20 address signal is not masked during this cycle but is masked in the next cycle. During a prefetch (cacheable or not), if A20M# is asserted after the first ADS#, A20 is not masked for the duration of the prefetch; even if BS16# or BS8# is asserted.

6.2.16 BOUNDARY SCAN TEST SIGNALS

The following boundary scan test signals are only available on the 50 MHz version of the Intel486 CPU.

Test Clock (TCK)

TCK is an input to the Intel486 CPU and provides the clocking function required by the JTAG boundary scan feature. TCK is used to clock state information and data into and out of the component. State select information and data are clocked into the component on the rising edge of TCK on TMS and TDI, respectively. Data is clocked out of the part on the falling edge of TCK on TDO.

In addition to using TCK as a free running clock, it may be stopped in a low, O, state, indefinitely as described in IEEE 1149.1. While TCK is stopped in the low state, the boundary scan latches retain their state.

When boundary scan is not used, TCK should be tied high or left as a NC (This is important during power up to avoid the possibility of glitches on the TCK which could prematurely initiate boundary scan operations). TCK is supplied with an internal pullup resistor.

TCK is a clock signal and is used as a reference for sampling other JTAG signals. On the rising edge of TCK, TMS and TDI are sampled. On the falling edge of TCK, TDO is driven.

Test Mode Select (TMS)

TMS is decoded by the JTAG TAP (Tap Access Port) to select the operation of the test logic, as described in Section 8.5.4.

To guarantee deterministic behavior of the TAP controller, TMS is provided with an internal pull-up resistor. If boundary scan is not used, TMS may be tied high or left unconnected. TMS is sampled on the rising edge of TCK. TMS is used to select the internal TAP states required to load boundary scan in-

structions to data on TDI. For proper initialization of the JTAG logic, TMS should be driven high, "1", for at least four TCK cycles following the rising edge of RESET.

Test Data Input (TDI)

TDI is the serial input used to shift JTAG instructions and data into the component. The shifting of instructions and data occurs during the SHIFT-IR and SHIFT-DR controller states, respectively. These states are selected using the TMS signal as described in Section 8.5.4.

An internal pull-up resistor is provided on TDI to ensure a known logic state if an open circuit occurs on the TDI path. Note that when "1" is continuously shifted into the instruction register, the BYPASS instruction is selected. TDI is sampled on the rising edge of TCK, during the SHIFT-IR and the SHIFT-DR states. During all other TAP controller states, TDI is a "don't care".

Test Data Output (TDO)

TDO is the serial output used to shift JTAG instructions and data out of the component. The shifting of instructions and data occurs during the SHIFT-IR and SHIFT-DR TAP controller states, respectively. These states are selected using the TMS signal as described in Section 8.5.4. When not in SHIFT-IR or SHIFT-DR state, TDO is driven to a high impedance state to allow connecting TDO of different devices in parallel.

TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT-DR TAP controller states. At all other times TDO is driven to the high impedance state.

6.3 Write Buffers

The Intel486 Microprocessor contains four write buffers to enhance the performance of consecutive writes to memory. The buffers can be filled at a rate of one write per clock until all four buffers are filled.

When all four buffers are empty and the bus is idle, a write request will propagate directly to the external bus bypassing the write buffers. If the bus is not available at the time the write is generated internally, the write will be placed in the write buffers and propagate to the bus as soon as the bus becomes available. The write is stored in the on-chip cache immediately if the write is a cache hit.

Writes will be driven onto the external bus in the same order in which they are received by the write buffers. Under certain conditions a memory read will go onto the external bus before the memory writes pending in the buffer even though the writes occurred earlier in the program execution.

A memory read will only be reordered in front of all writes in the buffers under the following conditions: If all writes pending in the buffers are cache hits and the read is a cache miss. Under these conditions the Intel486 Microprocessor will not read from an external memory location that needs to be updated by one of the pending writes.

Reordering of a read with the writes pending in the buffers can only occur once before all the buffers are emptied. Reordering read once only maintains cache consistency. Consider the following example:

The CPU writes to location X. Location X is in the internal cache, so it is updated there immediately. However, the bus is busy so the write out to main memory is buffered (see Figure 6.3(a)). At this point, any reads to location X would be cache hits and most up-to-date data would be read.

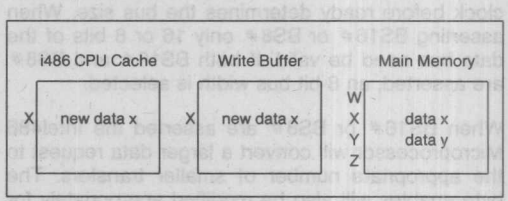


Figure 6.3(a)

The next instruction causes a read to location Y. Location Y is not in the cache (a cache miss). Since the write in the write buffer is a cache hit, the read is reordered. When location Y is read, it is put into the cache. The possibility exists that location Y will replace location X in the cache. If this is true, location X would no longer be cached (see Figure 6.3(b)).

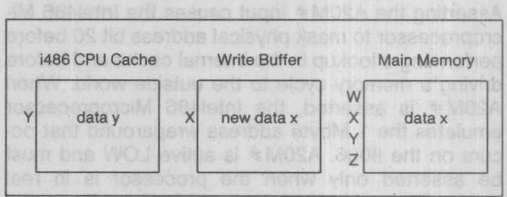


Figure 6.3(b)

Cache consistency has been maintained up to this point. If a subsequent read is to location X (now a cache miss) and it was reordered in front of the buffered write to location X, stale data would be read. This is why only 1 read is allowed to be reordered. Once a read is reordered, all the writes in the write buffer are flagged as cache misses to ensure that no more reads are reordered. Since one of the condi-

tions to reorder a read is that all writes in the write buffer must be cache hits, no more reordering is allowed until all of those flagged writes propogate to the bus. Similarly, if an invalidation cycle is run all entries in the write buffer are flagged as cache misses.

For multiple processor systems and/or systems using DMA techniques, such as bus snooping, locked semaphores should be used to maintain cache consistency.

6.3.1 WRITE BUFFERS AND I/O CYCLES

Input/Output (I/O) cycles must be handled in a different manner by the write buffers.

I/O reads are never reordered in front of buffered memory writes. This insures that the Intel486 Microprocessor will update all memory locations before reading status from an I/O device.

The Intel486 Microprocessor never buffers single I/O writes. When processing an OUT instruction, internal execution stops until the I/O write actually completes on the external bus. This allows time for the external system to drive an invalidate into the Intel486 Microprocessor or to mask interrupts before the processor progresses to the instruction following OUT. REP OUTS instructions will be buffered.

I/O device recovery time must be handled slightly differently by the Intel486 Microprocessor than with the 386 Microprocessor. I/O device back-to-back write recovery times could be guaranteed by the 386 Microprocessor by inserting a jump to the next instruction in the code that writes to the device. The jump forces the 386 Microprocessor to generate a prefetch bus cycle which can't begin until the I/O write completes.

Inserting a jump to the next write will not work with the Intel486 Microprocessor because the prefetch could be satisfied by the on-chip cache. A read cycle must be explicitly generated to a non-cacheable location in memory to guarantee that a read bus cycle is performed. This read will not be allowed to proceed to the bus until after the I/O write has completed because I/O writes are not buffered. The I/O device will have time to recover to accept another write during the read cycle.

6.3.2 WRITE BUFFERS IMPLICATIONS ON LOCKED BUS CYCLES

Locked bus cycles are used for read-modify-write accesses to memory. During a read-modify-write access, a memory base variable is read, modified and then written back to the same memory location. It is important that no other bus cycles, generated by other bus masters or by the Intel486 Microprocessor itself, be allowed on the external bus between the read and write portion of the locked sequence.

During a locked read cycle the Intel486 Microprocessor will always access external memory, it will never look for the location in the on-chip cache, but for write cycles, data is written in the internal cache (if cache hit) and in the external memory. All data pending in the Intel486 Microprocessor's write buffers will be written to memory before a locked cycle is allowed to proceed to the external bus.

The Intel486 Microprocessor will assert the LOCK# pin after the write buffers are emptied during a locked bus cycle. With the LOCK# pin asserted, the microprocessor will read the data, operate on the data and place the results in a write buffer. The contents of the write buffer will then be written to external memory. LOCK# will become inactive after the write part of the locked cycle.

6.4 Interrupt and Non-Maskable Interrupt Interface

The Intel486 Microprocessor provides two asynchronous interrupt inputs, INTR (interrupt request) and NMI (non-maskable interrupt input). This section describes the hardware interface between the instruction execution unit and the pins. For a description of the algorithmic response to interrupts refer to Section 2.7. For interrupt timings refer to Section 7.2.10.

6.4.1 INTERRUPT LOGIC

The Intel486 Microprocessor contains a two-clock synchronizer on the interrupt line. An interrupt request will reach the internal instruction execution unit two clocks after the INTR pin is asserted, if proper setup is provided to the first stage of the synchronizer.

There is no special logic in the interrupt path other than the synchronizer. The INTR signal is level sensitive and must remain active for the instruction execution unit to recognize it. The interrupt will not be serviced by the Intel486 Microprocessor if the INTR signal does not remain active.

The instruction execution unit will look at the state of the synchronized interrupt signal at specific clocks during the execution of instructions (if interrupts are enabled). These specific clocks are at instruction boundaries, or iteration boundaries in the case of string move instructions. Interrupts will only be accepted at these boundaries.

An interrupt must be presented to the Intel486 Microprocessor INTR pin three clocks before the end of an instruction for the interrupt to be acknowl-

edged. Presenting the interrupt 3 clocks before the end of an instruction allows the interrupt to pass through the two clock synchronizer leaving one clock to prevent the initiation of the next sequential instruction and to begin interrupt service. If the interrupt is not received in time to prevent the next instruction, it will be accepted at the end of next instruction, assuming INTR is still held active. The interrupt service microcode will start after two dead clocks.

The longest latency between when an interrupt request is presented on the INTR pin and when the interrupt service begins is: longest instruction used + the two clocks for synchronization + one clock required to vector into the interrupt service microcode.

6.4.2 NMI LOGIC

The NMI pin has a synchronizer like that used on the INTR line. Other than the synchronizer, the NMI logic is different from that of the maskable interrupt.

NMI is edge triggered as opposed to the level triggered INTR signal. The rising edge of the NMI signal is used to generate the interrupt request. The NMI input need not remain active until the interrupt is actually serviced. The NMI pin only needs to remain active for a single clock if the required setup and hold times are met. NMI will operate properly if it is held active for an arbitrary number of clocks.

The NMI input must be held inactive for at least four clocks after it is asserted to reset the edge triggered logic. A subsequent NMI may not be generated if the NMI is not held inactive for at least two clocks after being asserted.

The NMI input is internally masked whenever the NMI routine is entered. The NMI input will remain masked until an IRET (return from interrupt) instruction is executed. Masking the NMI signal prevents recursive NMI calls. If another NMI occurs while the NMI is masked off, the pending NMI will be executed after the current NMI is done. Only one NMI can be pending while NMI is masked.

6.5 Reset and Initialization

The Intel486 DX Microprocessor has a built in self test (BIST) that can be run during reset. The BIST is invoked if the AHOLD pin is asserted for 2 clocks before and 2 clocks after RESET is deasserted. RESET must be active for 15 clocks with or without BIST enabled. To ensure proper results, FLUSH# must not be asserted while BIST is executing. Refer to Section 8.0 for information on Intel486 DX Microprocessor testability.

The Intel486 Microprocessor registers have the values shown in Table 6.2 after RESET is performed. The EAX register contains information on the success or failure of the BIST if the self test is executed. The DX register always contains a component identifier at the conclusion of RESET. The upper byte of DX (DH) will contain 04 and the lower byte (DL) will contain a stepping identifier (see Table 6-3). The floating point registers are initialized as if the FINIT/FNINIT (initialize processor) instruction was executed if the BIST was performed. If the BIST is not executed, the floating point registers are unchanged.

Table 6.2. Register Values after Reset

Register	Initial Value (BIST)	Initial Value (No Bist)
EAX	Zero (Pass)	Undefined
ECX	Undefined	Undefined
EDX	0400 + Revision ID	0400 + Revision ID
EBX	Undefined	Undefined
ESP	Undefined	Undefined
EBP	Undefined	Undefined
ESI	Undefined	Undefined
EDI	Undefined	Undefined
EFLAGS	00000002h	00000002h
EIP	0FFF0h	0FFF0h
ES	0000h	0000h
CS	F000h*	F000h*
SS	0000h	0000h
DS	0000h	0000h
FS	0000h	0000h
GS	0000h	0000h
IDTR	Base = 0, Limit = 3FFh	Base = 0, Limit = 3FFh
CR0	60000010h	60000010h
DR7	00000000h	00000000h
CW	037Fh	Unchanged
SW	0000h	Unchanged
TW	FFFFh	Unchanged
FIP	00000000h	Unchanged
FEA	00000000h	Unchanged
FCS	0000h	Unchanged
FDS	0000h	Unchanged
FOP	000h	Unchanged
FSTACK	Undefined	Unchanged

Table 6-3. Intel486™ CPU Revision ID

Intel486™ CPU Stepping Name	Component ID	Revision ID
B3	04	01
B4	04	01
B5	04	01
B6	04	01
C0	04	02
C1	04	03
D0	04	04
cA2	04	10
cA3	04	10
cB0	04	11
cB1	04	11
Intel OverDrive™ Processor Stepping Name		
A2	04	32
B1	04	33

The Intel486 Microprocessor will start executing instructions at location FFFFFFF0H after RESET. When the first InterSegment Jump or Call is executed, address lines A20–A31 will drop LOW for CS-relative memory cycles, and the Intel486 Microprocessor will only execute instructions in the lower one Mbyte of physical memory. This allows the system designer to use a ROM at the top of physical memory to initialize the system and take care of RESETs.

RESET forces the Intel486 Microprocessor to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active.

All entries in the cache are invalidated by RESET.

6.5.1 PIN STATE DURING RESET

The Intel486 Microprocessor recognizes and can respond to HOLD, AHOLD, and BOFF# requests regardless of the state of RESET. Thus, even though the processor is in reset, it can still float its bus in response to any of these requests.

While in reset, the Intel486 Microprocessor bus is in the state shown in Figure 6.4 if the HOLD, AHOLD and BOFF# requests are inactive. Note that the address (A31–A2, BE3#–BE0#) and cycle definition (M/IO#, D/C#, W/R#) pins are undefined from the time reset is asserted up to the start of the first bus cycle. All undefined pins (except FERR#) assume known values at the beginning of the first bus cycle. The first bus cycle is always a code fetch to address FFFFFFF0H.

FERR# reflects the state of the ES (Error Summary status) bit in the floating point unit status word. The ES bit is initialized whenever the floating point unit state is initialized. The floating point unit's status word register can be initialized by BIST or by executing FINIT/FNINIT instruction. Thus, after reset and before executing the first FINIT or FNINIT instructon, the values of the FERR# and the numeric status word register bits 0–7 depends on whether or not BIST is performed. Table 6-4 shows the state of FERR# signal after reset and before the execution of the FINIT/FNINIT instruction.

Table 6-4

BIST Performed	FERR# Pin	FPU Status Word Register Bits 0–7
YES	Inactive (High)	Inactive (Low)
NO	Undefined (Low or High)	Undefined (Low or High)

After the first FINIT or FNINIT instruction, FERR# pin and the FPU status word register bits (0–7) will be inactive irrespective of the Built-In Self-Test (BIST).

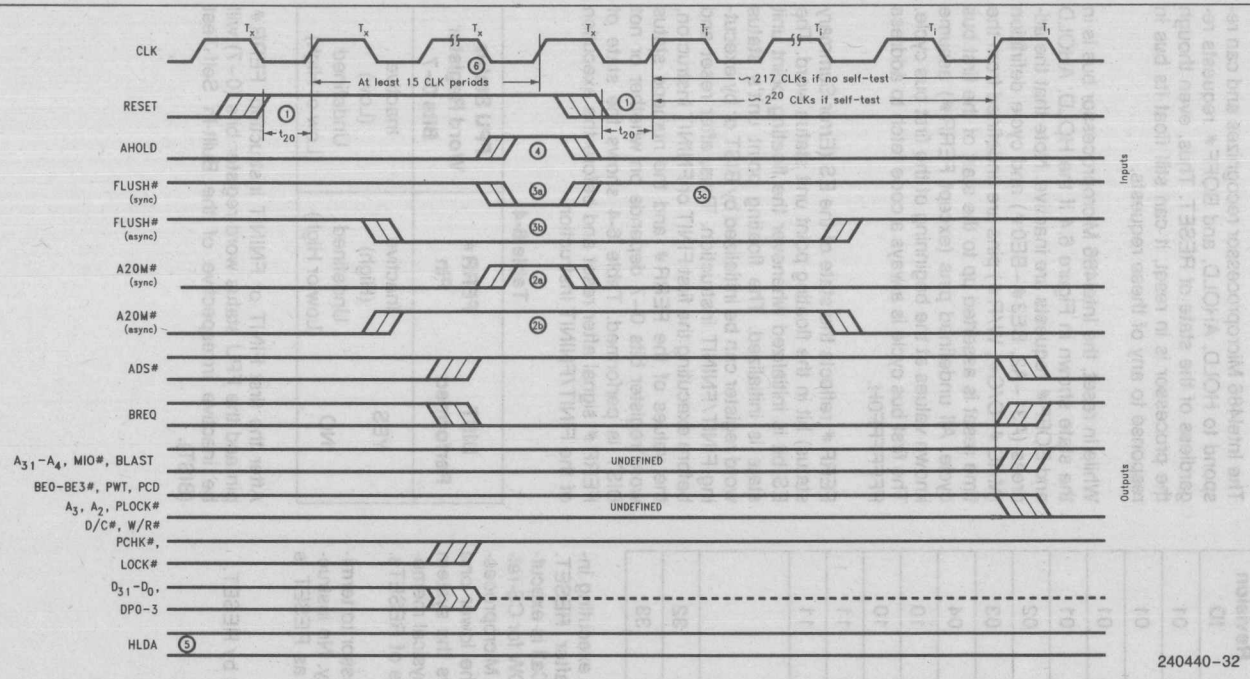


Figure 6.4. Pin States during RESET

NOTES:

1. RESET is an asynchronous input. t_{20} must be met only to guarantee recognition on a specific clock edge.
2a. When A20M# is driven synchronously, it must be driven high (inactive) for the CLK edge prior to the falling edge of RESET to ensure proper operation. A20M# setup and hold times must be met.
2b. When A20M# is driven asynchronously, it must be driven high (inactive) for two CLKs prior to and two CLKs after the falling edge of RESET to ensure proper operation.
3a. When FLUSH# is driven synchronously, it should be driven low (active) for the CLK edge prior to the falling edge of RESET to invoke the Tri-State Output Test Mode. All outputs are guaranteed tri-stated within 10 CLKs of RESET being deasserted. FLUSH# setup and hold times must be met.
3b. When FLUSH# is driven asynchronously, it must be driven low (active) for two CLKs prior to and two CLKs after the falling edge of RESET to invoke the Tri-State Output Test Mode. All outputs are guaranteed tri-stated within 10 CLKs of RESET being deasserted.
3c. FLUSH# must be driven high (inactive) during Build-in-Self-Test (BIST).
4. AHOLD should be driven high (active) for the CLK edge prior to the falling edge of RESET to invoke the Built-In-Self-Test (BIST). AHOLD setup and hold times must be met.
5. Hold is recognized normally during RESET.
6. 15 CLKs RESET pulse width for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.

7.0 BUS OPERATION

7.1 Data Transfer Mechanism

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte, word and dword lengths may be transferred without restrictions on physical address alignment. Data may be accessed at any byte boundary but two or three cycles may be required for unaligned data transfers. See Section 7.1.3 Dynamic Bus Sizing and 7.1.6 Operand Alignment.

The Intel486 Microprocessor address signals are split into two components. High-order address bits are provided by the address lines, A2–A31. The byte enables, BE0#–BE3#, form the low-order address and provide linear selects for the four bytes of the 32-bit address bus.

The byte enable outputs are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 7.1. Byte enable patterns which have a negated byte enable separating two or three asserted byte enables will never occur (see Table 7.5). All other byte enable patterns are possible.

Table 7.1. Byte Enables and Associated Data and Operand Bytes

Byte Enable Signal	Associated Data Bus Signals	
BE0#	D0–D7	(byte 0—least significant)
BE1#	D8–D15	(byte 1)
BE2#	D16–D23	(byte 2)
BE3#	D24–D31	(byte 3—most significant)

Address bits A0 and A1 of the physical operand's base address can be created when necessary. Use of the byte enables to create A0 and A1 is shown in Table 7.2. The byte enables can also be decoded to generate BLE# (byte low enable) and BHE# (byte high enable). These signals are needed to address 16-bit memory systems (see Section 7.1.4 Interfacing with 8- and 16-bit memories).

Table 7.2. Generating A0–A31 from BE0#–BE3# and A2–A31

Intel486™ CPU Address Signals								
A31	..... A2				BE3#	BE2#	BE1#	BE0#
	Physical Base Address							
A31	.....	A2	A1	A0				
A31	.....	A2	0	0	X	X	X	Low
A31	.....	A2	0	1	X	X	Low	High
A31	.....	A2	1	0	X	Low	High	High
A31	.....	A2	1	1	Low	High	High	High

7.1.1 MEMORY AND I/O SPACES

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. Physical memory addresses range from 00000000H to FFFFFFFFH (4 gigabytes). I/O addresses range from 00000000H to 0000FFFFH (64 Kbytes) for programmed I/O. See Figure 7.1.

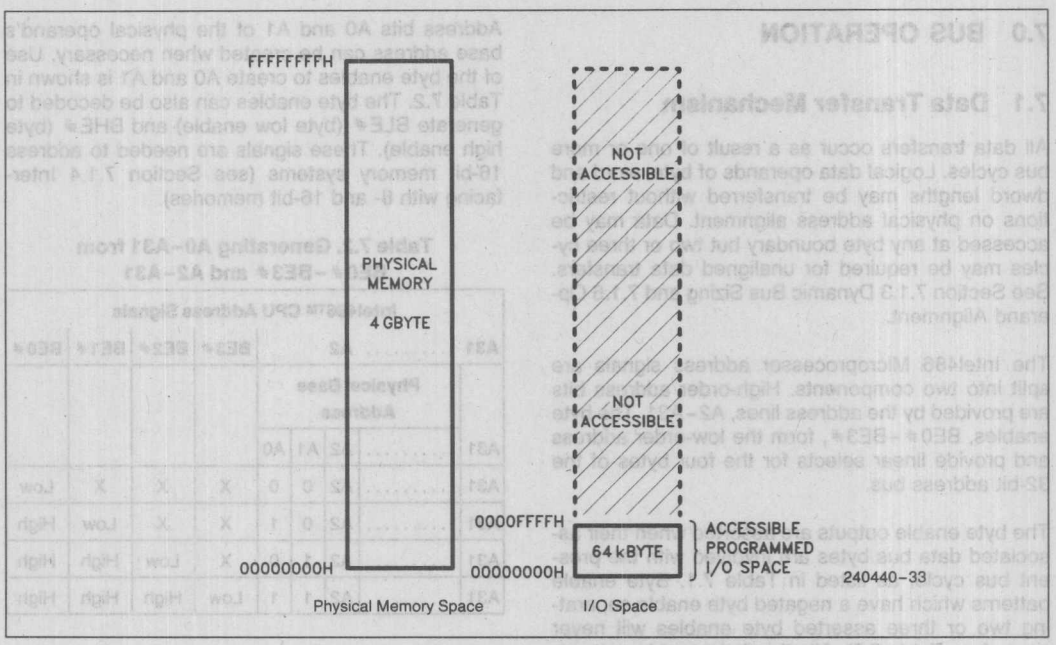


Figure 7.1. Physical Memory and I/O Spaces

7.1.2 MEMORY AND I/O SPACE ORGANIZATION

The Intel486 Microprocessor datapath to memory and input/output (I/O) spaces can be 32-, 16- or 8-bits wide. The byte enable signals, BE0#–BE3#, allow byte granularity when addressing any memory or I/O structure whether 8, 16 or 32 bits wide.

The Intel486 Microprocessor includes bus control pins, BS16# and BS8#, which allow direct connection to 16- and 8-bit memories and I/O devices. Cycles to 32-, 16- and 8-bit may occur in any sequence, since the BS8# and BS16# signals are sampled during each bus cycle.

32-bit wide memory and I/O spaces are organized as arrays of physical 4-byte words. Each memory or I/O 4-byte word has four individually addressable bytes at consecutive byte addresses (see Figure 7.2). The lowest addressed byte is associated with data signals D0–D7; the highest-addressed byte with D24–D31. Physical 4-byte words begin at addresses divisible by four.

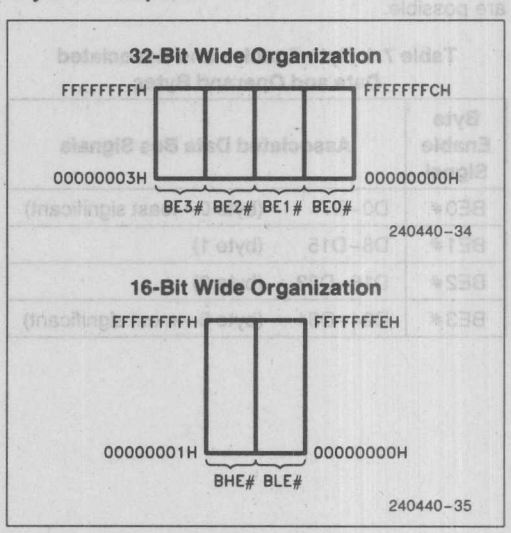


Figure 7.2. Physical Memory and I/O Space Organization

16-bit memories are organized as arrays of physical 2-byte words. Physical 2-byte words begin at addresses divisible by two. The byte enables BE0# – BE3#, must be decoded to A1, BLE# and BHE# to address 16-bit memories (see Section 7.1.4).

To address 8-bit memories, the two low order address bits A0 and A1, must be decoded from BE0# – BE3#. The same logic can be used for 8- and 16-bit memories since the decoding logic for BLE# and A0 are the same (see Section 7.1.4).

7.1.3 DYNAMIC DATA BUS SIZING

Dynamic data bus sizing is a feature allowing processor connection to 32-, 16- or 8-bit buses for memory or I/O. A processor may connect to all three bus sizes. Transfers to or from 32-, 16- or 8-bit devices are supported by dynamically determining the bus width during each bus cycle. Address decoding circuitry may assert BS16# for 16-bit devices, or BS8# for 8-bit devices during each bus cycle. BS8# and BS16# must be negated when addressing 32-bit devices. An 8-bit bus width is selected if both BS16# and BS8# are asserted.

BS16# and BS8# force the Intel486 Microprocessor to run additional bus cycles to complete requests larger than 16- or 8 bits. A 32-bit transfer will be converted into two 16-bit transfers (or 3 transfers if the data is misaligned) when BS16# is asserted. Asserting BS8# will convert a 32-bit transfer into four 8-bit transfers.

Extra cycles forced by BS16# or BS8# should be viewed as independent bus cycles. BS16# or BS8# must be driven active during each of the extra cycles unless the addressed device has the ability to change the number of bytes it can return between cycles.

The Intel486 Microprocessor will drive the byte enables appropriately during extra cycles forced by BS8# and BS16#. A2–A31 will not change if accesses are to a 32-bit aligned area. Table 7.3 shows the set of byte enables that will be generated on the next cycle for each of the valid possibilities of the byte enables on the current cycle.

The dynamic bus sizing feature of the Intel486 Microprocessor is significantly different than that of the 386 Microprocessor. Unlike the 386 Microprocessor, the Intel486 Microprocessor requires that data bytes be driven on the addressed data pins. The simplest example of this function is a 32-bit aligned, BS16# read. When the Intel486 Microprocessor reads the two high order bytes, they must be driven on the data bus pins D16–D31. The Intel486 Microprocessor expects the two low order bytes on D0–D15. The 386 Microprocessor expects both the high and low order bytes on D0–D15. The 386 Microprocessor always reads or writes data on the lower 16 bits of the data bus when BS16# is asserted.

The external system must contain buffers to enable the Intel486 Microprocessor to read and write data on the appropriate data bus pins. Table 7.4 shows the data bus lines where the Intel486 Microprocessor expects data to be returned for each valid combination of byte enables and bus sizing options.

Valid data will only be driven onto data bus pins corresponding to active byte enables during write cycles. Other pins in the data bus will be driven but they will not contain valid data. Unlike the 386 Microprocessor, the Intel486 Microprocessor will not duplicate write data onto parts of the data bus for which the corresponding byte enable is negated.

Table 7.3. Next Byte Enable Values for BSn# Cycles

Current				Next with BS8#				Next with BS16#			
BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#
1	1	1	0	n	n	n	n	n	n	n	n
1	1	0	0	1	1	0	1	n	n	n	n
1	0	0	0	1	0	0	1	1	0	1	1
0	0	0	0	0	0	0	1	0	0	1	1
1	1	0	1	n	n	n	n	n	n	n	n
1	0	0	1	1	0	1	1	1	0	1	1
0	0	0	1	0	0	1	1	0	0	1	1
1	0	1	1	n	n	n	n	n	n	n	n
0	0	1	1	0	1	1	1	n	n	n	n
0	1	1	1	n	n	n	n	n	n	n	n

"n" means that another bus cycle will not be required to satisfy the request.

Table 7.4. Data Pins Read with Different Bus Sizes

BE3#	BE2#	BE1#	BE0#	w/o BS8#/BS16#	w BS8#	W BS16#
1	1	1	0	D7–D0	D7–D0	D7–D0
1	1	0	0	D15–D0	D7–D0	D15–D0
1	0	0	0	D23–D0	D7–D0	D15–D0
0	0	0	0	D31–D0	D7–D0	D15–D0
1	1	0	1	D15–D8	D15–D8	D15–D8
1	0	0	1	D23–D8	D15–D8	D15–D8
0	0	0	1	D31–D8	D15–D8	D15–D8
1	0	1	1	D23–D16	D23–D16	D23–D16
0	0	1	1	D31–D16	D23–D16	D31–D16
0	1	1	1	D31–D24	D31–D24	D31–D24

7.1.4 INTERFACING WITH 8-, 16- AND 32-BIT MEMORIES

In 32-bit physical memories such as Figure 7.3, each 4-byte word begins at a byte address that is a multiple of four. A2–A31 are used as a 4-byte word select. BE0#–BE3# select individual bytes within the 4-byte word. BS8# and BS16# are negated for all bus cycles involving the 32-bit array.

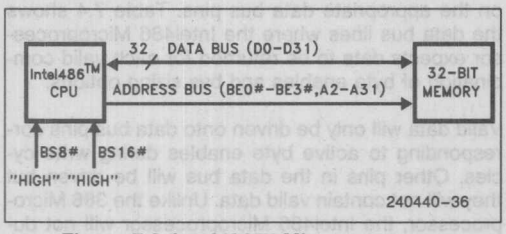


Figure 7.3. Intel486™ Microprocessor with 32-Bit Memory

16- and 8-bit memories require external byte swapping logic for routing data to the appropriate data lines and logic for generating BHE#, BLE# and A1. In systems where mixed memory widths are used, extra address decoding logic is necessary to assert BS16# or BS8#.

Figure 7.4 shows the Intel486 microprocessor address bus interface to 32-, 16- and 8-bit memories. To address 16-bit memories the byte enables must be decoded to produce A1, BHE# and BLE# (A0). For 8-bit wide memories the byte enables must be decoded to produce A0 and A1. The same byte select logic can be used in 16- and 8-bit systems since BLE# is exactly the same as A0 (see Table 7.5).

BE0#–BE3# can be decoded as shown in Table 7.5 to generate A1, BHE# and BLE#. The byte select logic necessary to generate BHE# and BLE# is shown in Figure 7.5.

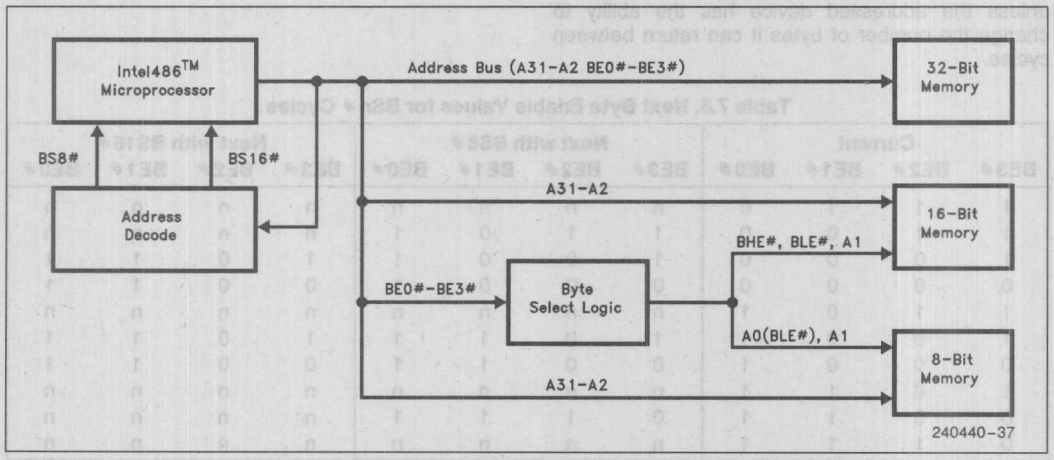


Figure 7.4. Addressing 16- and 8-Bit Memories

Table 7.5. Generating A1, BHE# and BLE# for Addressing 16-Bit Devices

Intel486™ CPU Signals				8, 16-Bit Bus Signals			Comments
BE3#	BE2#	BE1#	BE0#	A1	BHE#	BLE# (A0)	
H*	H*	H*	H*	x	x	x	x—no active bytes
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	
H*	L*	H*	L*	x	x	x	x—not contiguous bytes
H	L	L	H	L	L	H	
H	L	L	L	L	L	L	
L	H	H	H	H	L	H	
L*	H*	H*	L*	x	x	x	x—not contiguous bytes
L*	H*	L*	H*	x	x	x	x—not contiguous bytes
L*	H*	L*	L*	x	x	x	x—not contiguous bytes
L	L	H	H	H	L	L	
L*	L*	H*	L*	x	x	x	x—not contiguous bytes
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

BLE# asserted when D0–D7 of 16-bit bus is active.
BHE# asserted when D8–D15 of 16-bit bus is active.
A1 low for all even words; A1 high for all odd words.

Key:
x = don't care
H = high voltage level
L = low voltage level
* = a non-occurring pattern of Byte Enables; either none are asserted, or the pattern has Byte Enables asserted for non-contiguous bytes

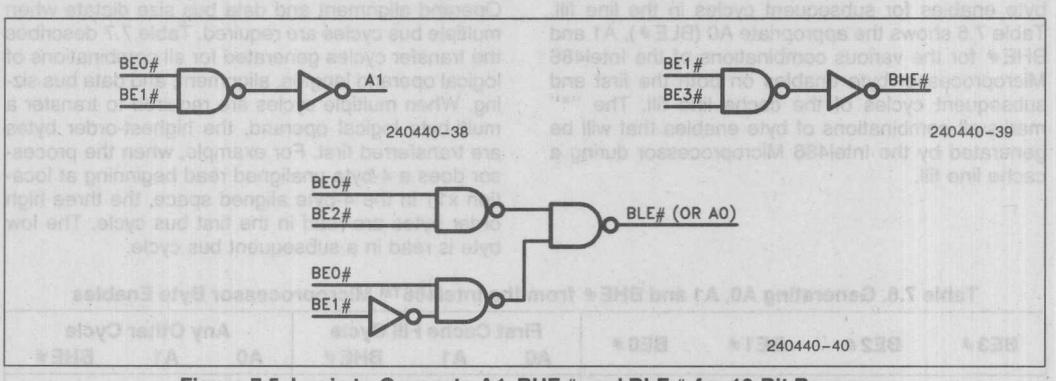


Figure 7.5. Logic to Generate A1, BHE# and BLE# for 16-Bit Busses

Combinations of BE0#–BE3# which never occur are those in which two or three asserted byte enables are separated by one or more negated byte enables. These combinations are "don't care" conditions in the decoder. A decoder can use the non-occurring BE0#–BE3# combinations to its best advantage.

Figure 7.6 shows an Intel486 Microprocessor data bus interface to 16- and 8-bit wide memories. External byte swapping logic is needed on the data lines so that data is supplied to, and received from the Intel486 Microprocessor on the correct data pins (see Table 7.4).

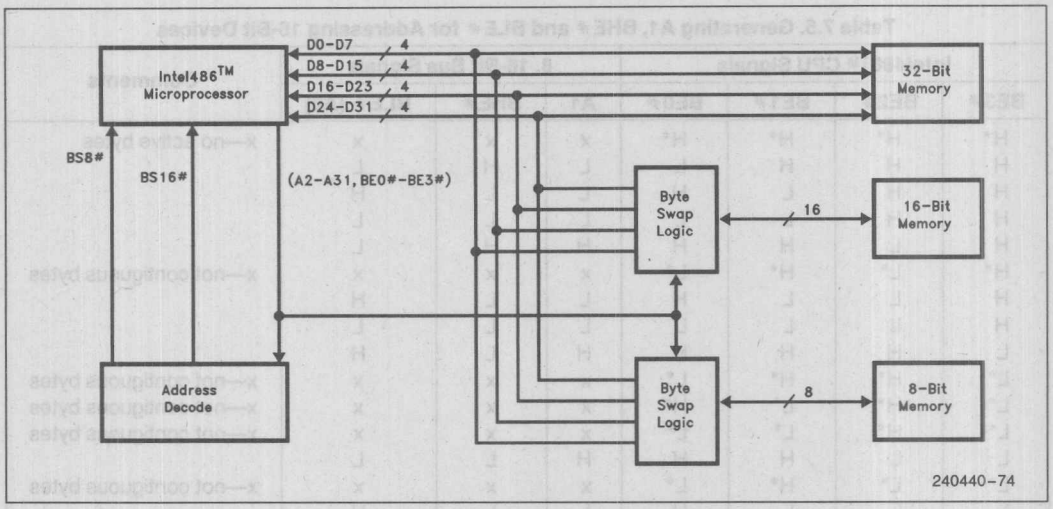


Figure 7.6. Data Bus Interface to 16- and 8-bit Memories

7.1.5 DYNAMIC BUS SIZING DURING CACHE LINE FILLS

BS8# and BS16# can be driven during cache line fills. The Intel486 Microprocessor will generate enough 8- or 16-bit cycles to fill the cache line. This can be up to 16 8-bit cycles.

The external system should assume that all byte enables are active for the first cycle of a cache line fill. The Intel486 Microprocessor will generate proper byte enables for subsequent cycles in the line fill. Table 7.6 shows the appropriate A0 (BLE#), A1 and BHE# for the various combinations of the Intel486 Microprocessor byte enables on both the first and subsequent cycles of the cache line fill. The "*" marks all combinations of byte enables that will be generated by the Intel486 Microprocessor during a cache line fill.

7.1.6 OPERAND ALIGNMENT

Physical 4-byte words begin at addresses that are multiples of four. It is possible to transfer a logical operand that spans more than one physical 4-byte word of memory or I/O at the expense of extra cycles. Examples are 4-byte operands beginning at addresses that are not evenly divisible by 4, or 2-byte words split between two physical 4-byte words. These are referred to as unaligned transfers.

Operand alignment and data bus size dictate when multiple bus cycles are required. Table 7.7 describes the transfer cycles generated for all combinations of logical operand lengths, alignment, and data bus sizing. When multiple cycles are required to transfer a multi-byte logical operand, the highest-order bytes are transferred first. For example, when the processor does a 4-byte unaligned read beginning at location x11 in the 4-byte aligned space, the three high order bytes are read in the first bus cycle. The low byte is read in a subsequent bus cycle.

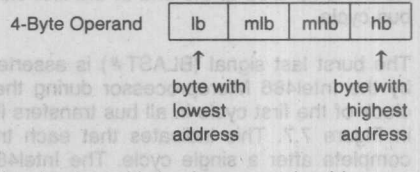
Table 7.6. Generating A0, A1 and BHE# from the Intel486™ Microprocessor Byte Enables

BE3#	BE2#	BE1#	BE0#	First Cache Fill Cycle			Any Other Cycle		
				A0	A1	BHE#	A0	A1	BHE#
1	1	1	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
*0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	1	0	0
1	0	0	1	0	0	0	1	0	0
*0	0	0	1	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	1
*0	0	1	1	0	0	0	0	1	0
*0	1	1	1	0	0	0	1	1	0

Table 7.7. Transfer Bus Cycles for Bytes, Words and Dwords

	Byte-Length of Logical Operand								
	1	2				4			
Physical Byte Address in Memory (Low Order Bits)	xx	00	01	10	11	00	01	10	11
Transfer Cycles over 32-Bit Bus	b	w	w	w	hb lb	d	hb l3	hw lw	h3 lb
Transfer Cycles over 16-Bit Data Bus = BS16# Asserted	b	w	lb hb	w	hb lb	lw hw	hb lb mw	hw lw	mw hb lb
Transfer Cycles over 8-Bit Data Bus = BS8# Asserted	b	lb hb	lb hb	lb hb	hb lb	lb mlb mhb hb	hb lb mlb mhb	mhb hb lb mlb	mlb mhb hb lb

KEY:
b = byte transfer
w = 2-byte transfer
3 = 3-byte transfer
d = 4-byte transfer
h = high-order portion
l = low-order portion
m = mid-order portion



The function of unaligned transfers with dynamic bus sizing is not obvious. When the external systems asserts BS16# or BS8# forcing extra cycles, low-order bytes or words are transferred first (opposite to the example above). When the Intel486 Microprocessor requests a 4-byte read and the external system asserts BS16#, the lower 2 bytes are read first followed by the upper 2 bytes.

In the unaligned transfer described above, the processor requested three bytes on the first cycle. If the external system asserted BS16# during this 3-byte transfer, the lower word is transferred first followed by the upper byte. In the final cycle the lower byte of the 4-byte operand is transferred as in the 32-bit example above.

7.2 Bus Functional Description

The Intel486 Microprocessor supports a wide variety of bus transfers to meet the needs of high performance systems. Bus transfers can be single cycle or multiple cycle, burst or non-burst, cacheable or non-cacheable, 8-, 16- or 32-bit, and pseudo-locked. To support multiprocessing systems there are cache invalidation cycles and locked cycles.

This section begins with basic non-cacheable non-burst single cycle transfers. It moves on to multiple cycle transfers and introduces the burst mode. Cacheability is introduced in Section 7.2.3. The remaining sections describe locked, pseudo-locked, invalidate, bus hold and interrupt cycles.

Bus cycles and data cycles are discussed in this section. A bus cycle is at least two clocks long and begins with ADS# active in the first clock and ready active in the last clock. Data is transferred to or from the Intel486 Microprocessor during a data cycle. A bus cycle contains one or more data cycles.

Refer to Section 7.2.13 for a description of the bus states shown in the timing diagrams.

7.2.1 NON-CACHEABLE NON-BURST SINGLE CYCLE

7.2.1.1 No Wait States

The fastest non-burst bus cycle that the Intel486 Microprocessor supports is two clocks long. These cycles are called 2-2 cycles because reads and writes take two cycles each. The first 2 refers to reads and

the second to writes. For example, if a wait state needs to be added to a write, the cycle would be called 2-3.

Basic two clock read and write cycles are shown in Figure 7.7. The Intel486 Microprocessor initiates a cycle by asserting the address status signal (ADS#) at the rising edge of the first clock. The ADS# output indicates that a valid bus cycle definition and address is available on the cycle definition lines and address bus.

The non-burst ready input (RDY#) is returned by the external system in the second clock. RDY# indicates that the external system has presented valid data on the data pins in response to a read or the external system has accepted data in response to a write.

The Intel486 Microprocessor samples RDY# at the end of the second clock. The cycle is complete if RDY# is active (LOW) when sampled. Note that RDY# is ignored at the end of the first clock of the bus cycle.

The burst last signal (BLAST#) is asserted (LOW) by the Intel486 Microprocessor during the second clock of the first cycle in all bus transfers illustrated in Figure 7.7. This indicates that each transfer is complete after a single cycle. The Intel486 Microprocessor asserts BLAST# in the last cycle of a bus transfer.

The timing of the parity check output (PCHK#) is shown in Figure 7.7. The Intel486 Microprocessor drives the PCHK# output one clock after ready terminates a read cycle. PCHK# indicates the parity status for the data sampled at the end of the previous clock. The PCHK# signal can be used by the external system. The Intel486 Microprocessor does nothing in response to the PCHK# output.

7.2.1.2 Inserting Wait States

The external system can insert wait states into the basic 2-2 cycle by driving RDY# inactive at the end of the second clock. RDY# must be driven inactive to insert a wait state. Figure 7.8 illustrates a simple non-burst, non-cacheable signal with one wait state added. Any number of wait states can be added to an Intel486 Microprocessor bus cycle by maintaining RDY# inactive.

The burst ready input (BRDY#) must be driven inactive on all clock edges where RDY# is driven inactive for proper operation of these simple non-burst cycles.

7.2.2 MULTIPLE AND BURST CYCLE BUS TRANSFERS

Multiple cycle bus transfers can be caused by internal requests from the Intel486 Microprocessor or by the external memory system. An internal request for a 64-bit floating point load or a 128-bit pre-fetch must take more than one cycle. Internal requests for unaligned data may also require multiple bus cycles. A cache line fill requires multiple cycles to complete. The external system can cause a multiple cycle transfer when it can only supply 8 or 16 bits per cycle.

Only multiple cycle transfers caused by internal requests are considered in this section. Cacheable cycles and 8- and 16-bit transfers are covered in Sections 7.2.3 and 7.2.5.

7.2.2.1 Burst Cycles

The Intel486 Microprocessor can accept burst cycles for any bus requests that require more than a single data cycle. During burst cycles, a new data item is strobed into the Intel486 Microprocessor every clock rather than every other clock as in non-burst cycles. The fastest burst cycle requires 2 clocks for the first data item with subsequent data items returned every clock.

The Intel486 Microprocessor is capable of bursting a maximum of 32 bits during a write. Burst writes can only occur if BS8# or BS16# is asserted. For example, the Intel486 Microprocessor can burst write four 8-bit operands or two 16-bit operands in a single burst cycle. But the Intel486 Microprocessor cannot burst multiple 32-bit writes in a single burst cycle.

Burst cycles begin with the Intel486 Microprocessor driving out an address and asserting ADS# in the same manner as non-burst cycles. The Intel486 microprocessor indicates that it is willing to perform a burst cycle by holding the burst last signal (BLAST#) inactive in the second clock of the cycle. The external system indicates its willingness to do a burst cycle by returning the burst ready signal (BRDY#) active.

The addresses of the data items in a burst cycle will all fall within the same 16-byte aligned area (corresponding to an internal Intel486 Microprocessor cache line). A 16-byte aligned area begins at location XXXXXXX0 and ends at location XXXXXXXF. During a burst cycle, only BE0–3#, A_2, and A_3 may change. A_4–A_{31}, M/IO#, D/C#, and W/R# will remain stable throughout a burst. Given the first address in a burst, external hardware can easily calculate the address of subsequent transfers in advance. An external memory system can be designed to quickly fill the Intel486 microprocessor internal cache lines.

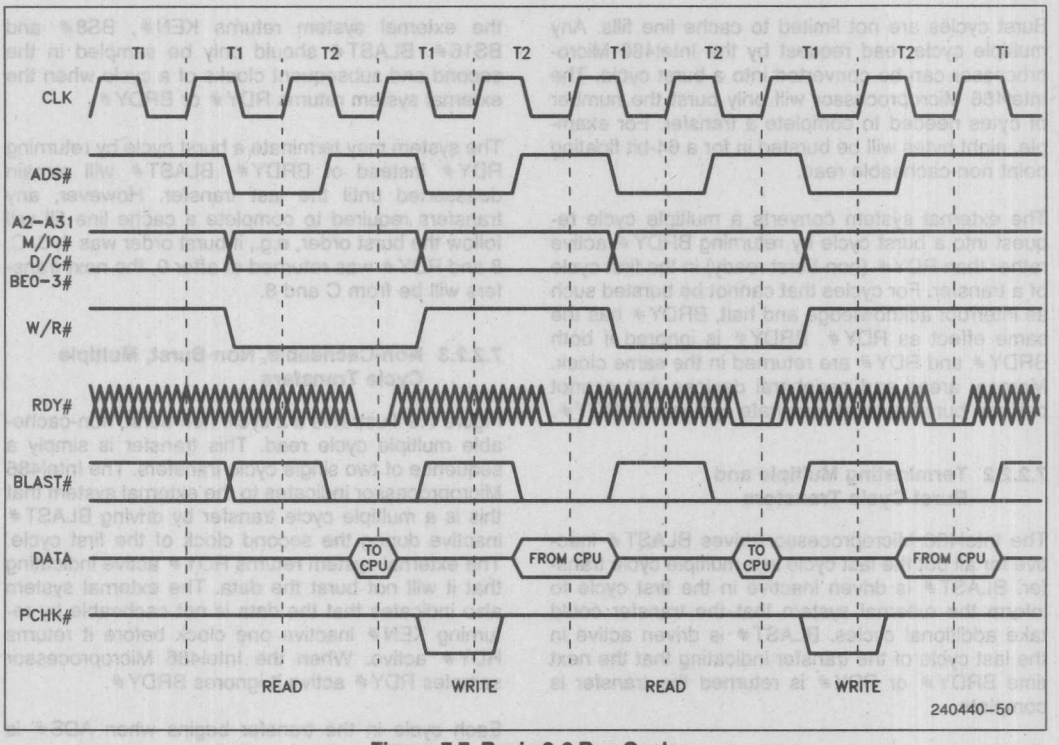


Figure 7.7. Basic 2-2 Bus Cycle

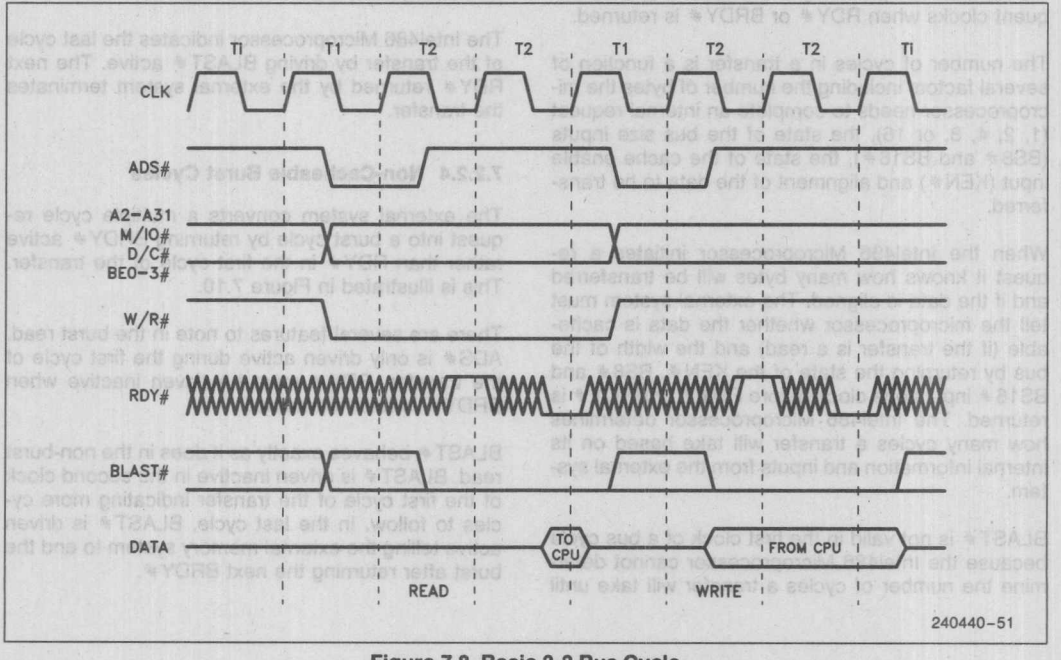


Figure 7.8. Basic 3-3 Bus Cycle

Burst cycles are not limited to cache line fills. Any multiple cycle read request by the Intel486 Microprocessor can be converted into a burst cycle. The Intel486 Microprocessor will only burst the number of bytes needed to complete a transfer. For example, eight bytes will be bursted in for a 64-bit floating point non-cacheable read.

The external system converts a multiple cycle request into a burst cycle by returning BRDY# active rather than RDY# (non-burst ready) in the first cycle of a transfer. For cycles that cannot be bursted such as interrupt acknowledge and halt, BRDY# has the same effect as RDY#. BRDY# is ignored if both BRDY# and RDY# are returned in the same clock. Memory areas and peripheral devices that cannot perform bursting must terminate cycles with RDY#.

7.2.2.2 Terminating Multiple and Burst Cycle Transfers

The Intel486 Microprocessor drives BLAST# inactive for all but the last cycle in a multiple cycle transfer. BLAST# is driven inactive in the first cycle to inform the external system that the transfer could take additional cycles. BLAST# is driven active in the last cycle of the transfer indicating that the next time BRDY# or RDY# is returned the transfer is complete.

BLAST# is not valid in the first clock of a bus cycle. It should be sampled only in the second and subsequent clocks when RDY# or BRDY# is returned.

The number of cycles in a transfer is a function of several factors including the number of bytes the microprocessor needs to complete an internal request (1, 2, 4, 8, or 16), the state of the bus size inputs (BS8# and BS16#), the state of the cache enable input (KEN#) and alignment of the data to be transferred.

When the Intel486 Microprocessor initiates a request it knows how many bytes will be transferred and if the data is aligned. The external system must tell the microprocessor whether the data is cacheable (if the transfer is a read) and the width of the bus by returning the state of the KEN#, BS8# and BS16# inputs one clock before RDY# or BRDY# is returned. The Intel486 Microprocessor determines how many cycles a transfer will take based on its internal information and inputs from the external system.

BLAST# is not valid in the first clock of a bus cycle because the Intel486 Microprocessor cannot determine the number of cycles a transfer will take until the external system returns KEN#, BS8# and BS16#. BLAST# should only be sampled in the second and subsequent clocks of a cycle when the external system returns RDY# or BRDY#.

The system may terminate a burst cycle by returning RDY# instead of BRDY#. BLAST# will remain deasserted until the last transfer. However, any transfers required to complete a cache line fill will follow the burst order, e.g., if burst order was 4, 0, C, 8 and RDY# was returned at after 0, the next transfers will be from C and 8.

7.2.2.3 Non-Cacheable, Non-Burst, Multiple Cycle Transfers

Figure 7.9 illustrates a 2 cycle non-burst, non-cacheable multiple cycle read. This transfer is simply a sequence of two single cycle transfers. The Intel486 Microprocessor indicates to the external system that this is a multiple cycle transfer by driving BLAST# inactive during the second clock of the first cycle. The external system returns RDY# active indicating that it will not burst the data. The external system also indicates that the data is not cacheable by returning KEN# inactive one clock before it returns RDY# active. When the Intel486 Microprocessor samples RDY# active it ignores BRDY#.

Each cycle in the transfer begins when ADS# is driven active and the cycle is complete when the external system returns RDY# active.

The Intel486 Microprocessor indicates the last cycle of the transfer by driving BLAST# active. The next RDY# returned by the external system terminates the transfer.

7.2.2.4 Non-Cacheable Burst Cycles

The external system converts a multiple cycle request into a burst cycle by returning BRDY# active rather than RDY# in the first cycle of the transfer. This is illustrated in Figure 7.10.

There are several features to note in the burst read. ADS# is only driven active during the first cycle of the transfer. RDY# must be driven inactive when BRDY# is returned active.

BLAST# behaves exactly as it does in the non-burst read. BLAST# is driven inactive in the second clock of the first cycle of the transfer indicating more cycles to follow. In the last cycle, BLAST# is driven active telling the external memory system to end the burst after returning the next BRDY#.

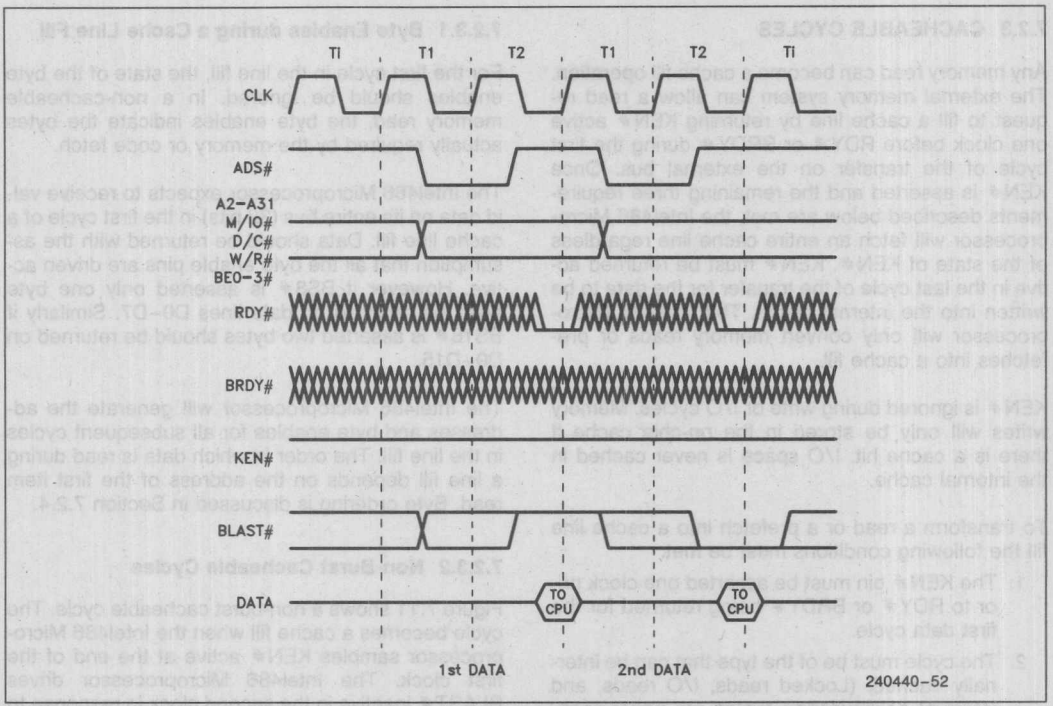


Figure 7.9. Non-Cacheable, Non-Burst, Multiple Cycle Transfers

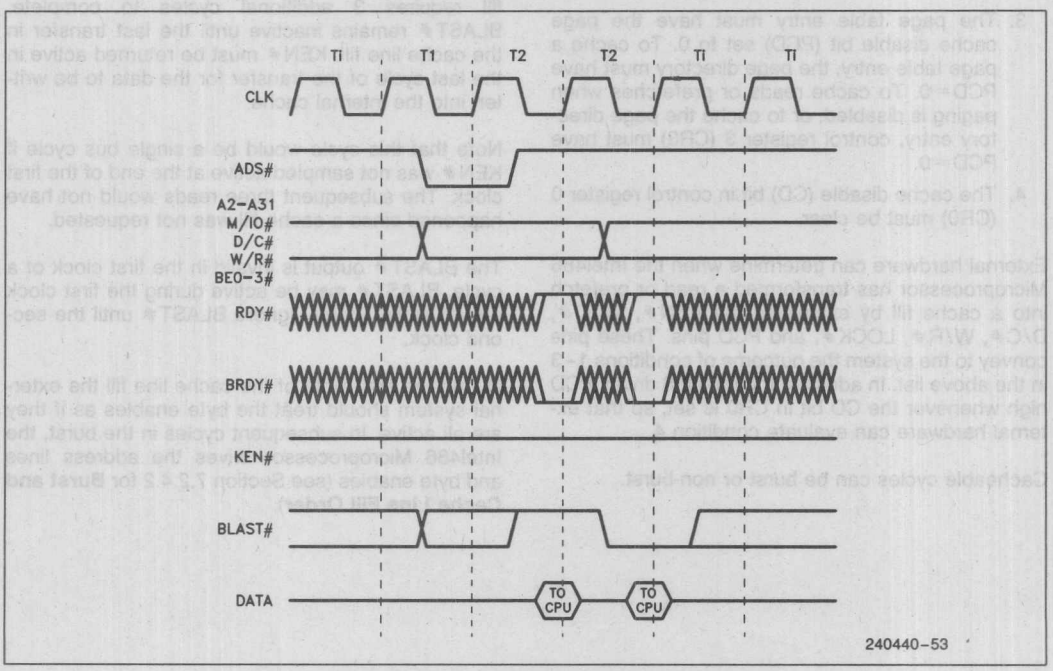


Figure 7.10. Non-Cacheable Burst Cycle

7.2.3 CACHEABLE CYCLES

Any memory read can become a cache fill operation. The external memory system can allow a read request to fill a cache line by returning KEN# active one clock before RDY# or BRDY# during the first cycle of the transfer on the external bus. Once KEN# is asserted and the remaining three requirements described below are met, the Intel486 Microprocessor will fetch an entire cache line regardless of the state of KEN#. KEN# must be returned active in the last cycle of the transfer for the data to be written into the internal cache. The Intel486 Microprocessor will only convert memory reads or prefetches into a cache fill.

KEN# is ignored during write or I/O cycles. Memory writes will only be stored in the on-chip cache if there is a cache hit. I/O space is never cached in the internal cache.

To transform a read or a prefetch into a cache line fill the following conditions must be met:

1. The KEN# pin must be asserted one clock prior to RDY# or BRDY# being returned for the first data cycle.
2. The cycle must be of the type that can be internally cached. (Locked reads, I/O reads, and interrupt acknowledge cycles are never cached).
3. The page table entry must have the page cache disable bit (PCD) set to 0. To cache a page table entry, the page directory must have PCD = 0. To cache reads or prefetches when paging is disabled, or to cache the page directory entry, control register 3 (CR3) must have PCD = 0.
4. The cache disable (CD) bit in control register 0 (CR0) must be clear.

External hardware can determine when the Intel486 Microprocessor has transformed a read or prefetch into a cache fill by examining the KEN#, M/IO#, D/C#, W/R#, LOCK#, and PCD pins. These pins convey to the system the outcome of conditions 1–3 in the above list. In addition, the Intel486 drives PCD high whenever the CD bit in CR0 is set, so that external hardware can evaluate condition 4.

Cacheable cycles can be burst or non-burst.

7.2.3.1 Byte Enables during a Cache Line Fill

For the first cycle in the line fill, the state of the byte enables should be ignored. In a non-cacheable memory read, the byte enables indicate the bytes actually required by the memory or code fetch.

The Intel486 Microprocessor expects to receive valid data on its entire bus (32 bits) in the first cycle of a cache line fill. Data should be returned with the assumption that all the byte enable pins are driven active. However if BS8# is asserted only one byte need be returned on data lines D0–D7. Similarly if BS16# is asserted two bytes should be returned on D0–D15.

The Intel486 Microprocessor will generate the addresses and byte enables for all subsequent cycles in the line fill. The order in which data is read during a line fill depends on the address of the first item read. Byte ordering is discussed in Section 7.2.4.

7.2.3.2 Non-Burst Cacheable Cycles

Figure 7.11 shows a non-burst cacheable cycle. The cycle becomes a cache fill when the Intel486 Microprocessor samples KEN# active at the end of the first clock. The Intel486 Microprocessor drives BLAST# inactive in the second clock in response to KEN#. BLAST# is driven inactive because a cache fill requires 3 additional cycles to complete. BLAST# remains inactive until the last transfer in the cache line fill. KEN# must be returned active in the last cycle of the transfer for the data to be written into the internal cache.

Note that this cycle would be a single bus cycle if KEN# was not sampled active at the end of the first clock. The subsequent three reads would not have happened since a cache fill was not requested.

The BLAST# output is invalid in the first clock of a cycle. BLAST# may be active during the first clock due to earlier inputs. Ignore BLAST# until the second clock.

During the first cycle of the cache line fill the external system should treat the byte enables as if they are all active. In subsequent cycles in the burst, the Intel486 Microprocessor drives the address lines and byte enables (see Section 7.2.4.2 for **Burst and Cache Line Fill Order**).

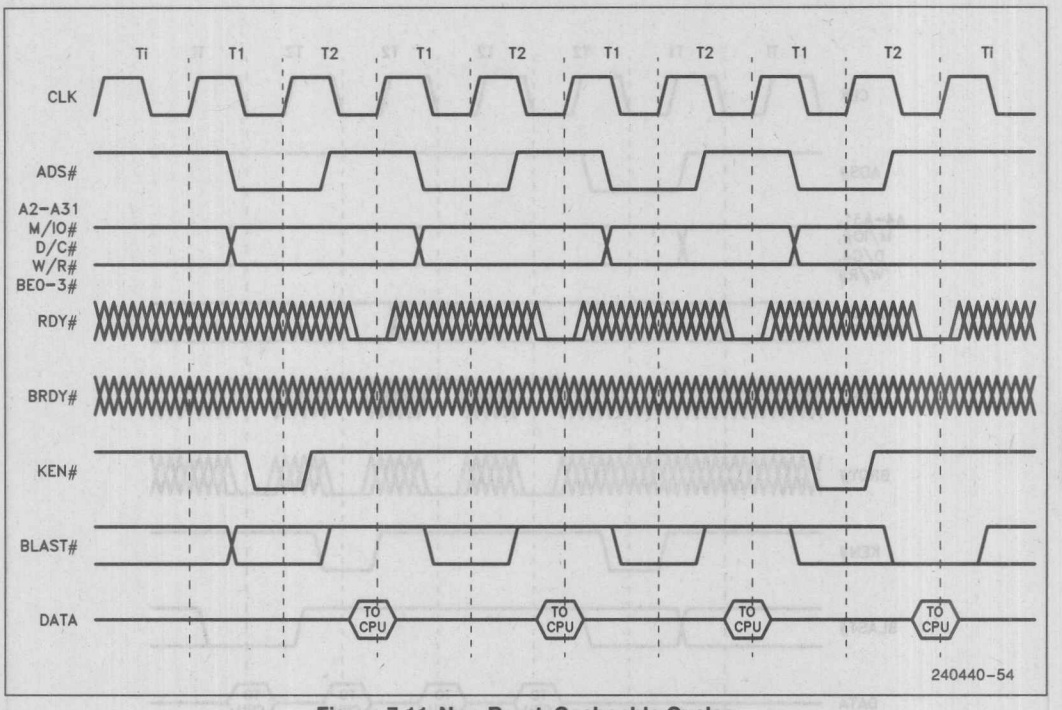


Figure 7.11. Non-Burst, Cacheable Cycles

7.2.3.3 Burst Cacheable Cycles

Figure 7.12 illustrates a burst mode cache fill. As in Figure 7.11, the transfer becomes a cache line fill when the external system returns KEN# active at the end of the first clock in the cycle.

The external system informs the Intel486 Microprocessor that it will burst the line in by driving BRDY# active at the end of the first cycle in the transfer.

Note that during a burst cycle ADS# is only driven with the first address.

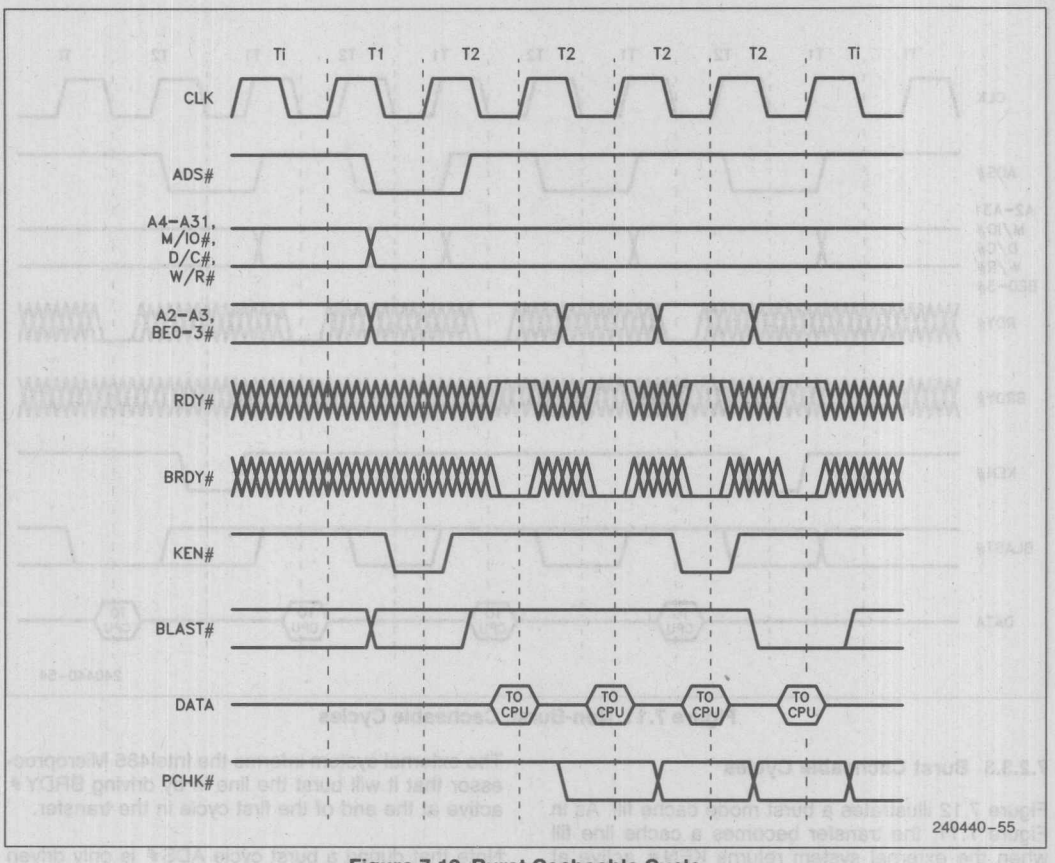


Figure 7.12. Burst Cacheable Cycle

7.2.3.4 Effect of Changing KEN# during a Cache Line Fill

KEN# can change multiple times as long as it arrives at its final value in the clock before RDY# or BRDY# is returned. This is illustrated in Figure 7.13. Note that the timing of BLAST# follows that of KEN# by one clock. The Intel486 samples KEN# every clock and uses the value returned in the clock before ready to determine if a bus cycle would be a cache line fill. Similarly, it uses the value of KEN# in the last cycle, before early RDY# to load the line just retrieved from the memory into the cache. KEN# is sampled every clock, it must satisfy setup and hold time.

KEN# can also change multiple times before a burst cycle as long as it arrives at its final value one clock before ready is returned active.

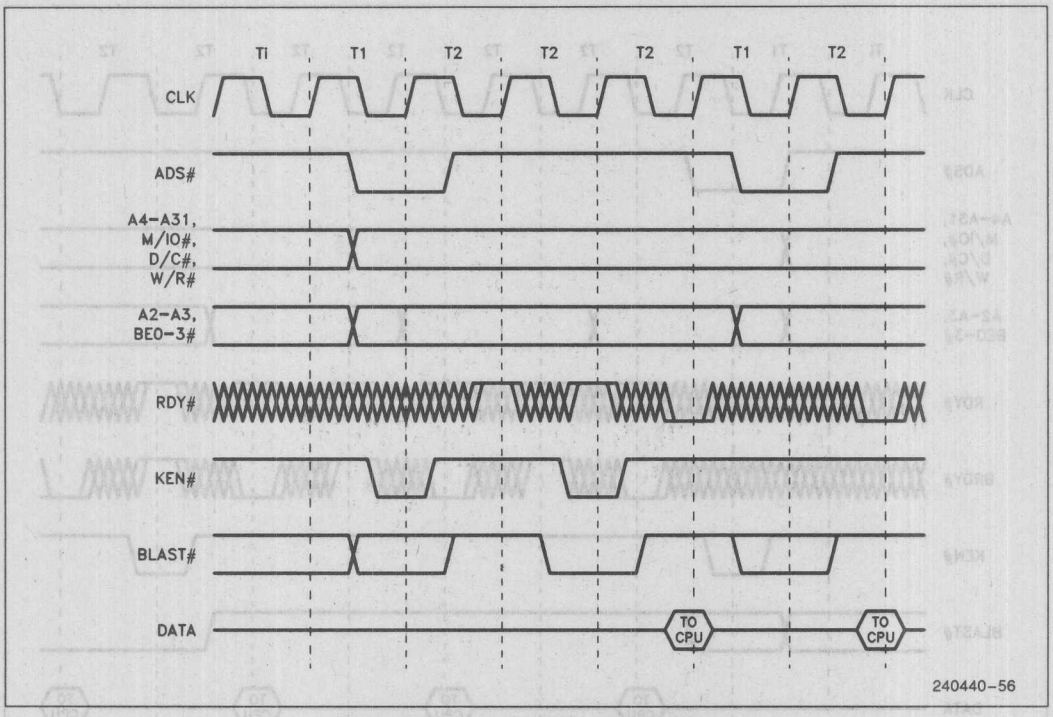


Figure 7.13. Effect of Changing KEN#

7.2.4 BURST MODE DETAILS

7.2.4.1 Adding Wait States to Burst Cycles

Burst cycles need not return data on every clock. The Intel486 Microprocessor will only strobe data into the chip when either RDY# or BRDY# are active.

Driving BRDY# and RDY# inactive adds a wait state to the transfer. A burst cycle where two clocks are required for every burst item is shown in Figure 7.14.

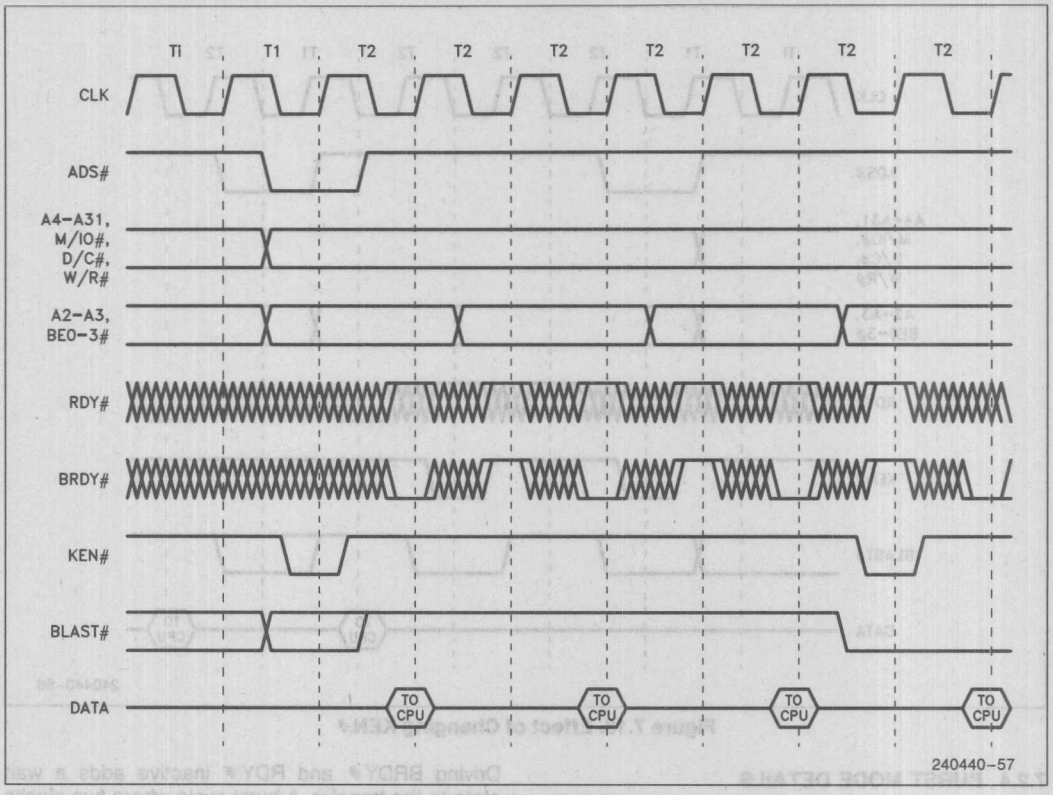


Figure 7.14. Slow Burst Cycle

7.2.4.2 Burst and Cache Line Fill Order

The burst order used by the Intel486 Microprocessor is shown in Table 7.8. This burst order is followed by any burst cycle (cache or not), cache line fill (burst or not) or code prefetch.

The microprocessor presents each request for data in an order determined by the first address in the transfer. For example, if the first address was 104 the next three addresses in the burst will be 100, 10C and 108.

Table 7.8. Burst Order

First Addr.	Second Addr.	Third Addr.	Fourth Addr.
0	4	8	C
4	0	C	8
8	C	0	4
C	8	4	0

An example of burst address sequencing is shown in Figure 7.15.

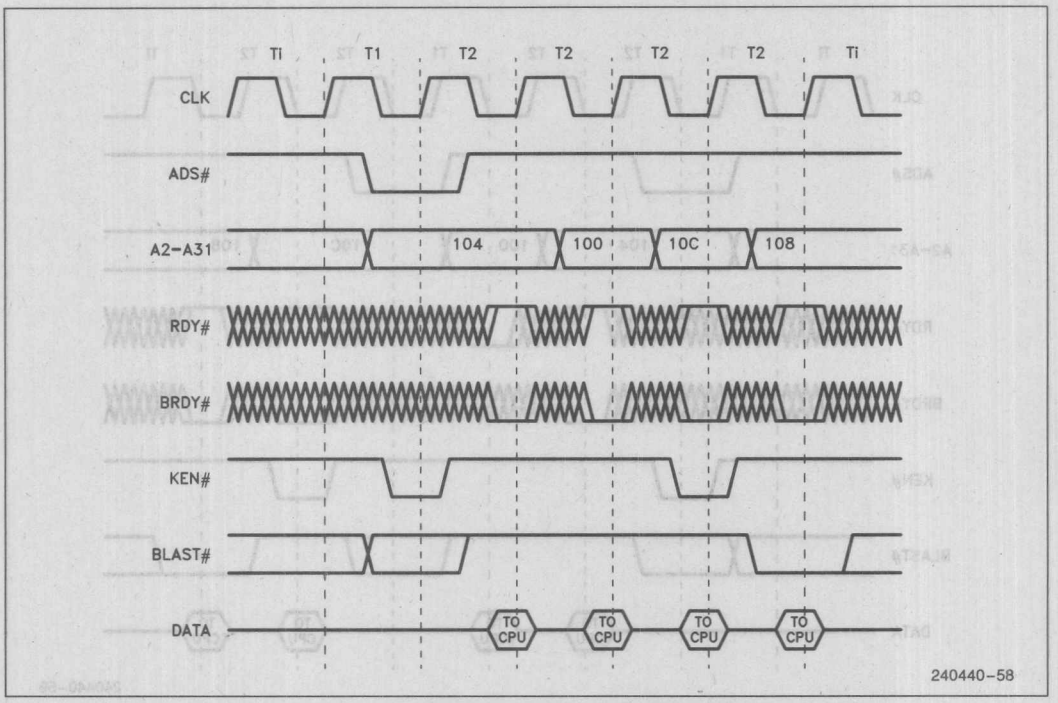


Figure 7.15. Burst Cycle Showing Order of Addresses

The sequences shown in Table 7.7 accommodate systems with 64-bit busses as well as systems with 32-bit data busses. The sequence applies to all bursts, regardless of whether the purpose of the burst is to fill a cache line, do a 64-bit read, or do a pre-fetch. If either BS8# or BS16# is returned active, the Intel486 Microprocessor completes the transfer of the current 32-bit word before progressing to the next 32-bit word. For example, a BS16# burst to address 4 has the following order: 4-6-0-2-C-E-8-A.

7.2.4.3 Interrupted Burst Cycles

Some memory systems may not be able to respond with burst cycles in the order defined in Table 7.7. To support these systems the Intel486 Microprocessor allows a burst cycle to be interrupted at any time. The Intel486 Microprocessor will automatically generate another normal bus cycle after being interrupted to complete the data transfer. This is called an interrupted burst cycle. The external system can respond to an interrupted burst cycle with another burst cycle.

The external system can interrupt a burst cycle by returning RDY# instead of BRDY#. RDY# can be returned after any number of data cycles terminated with BRDY#.

An example of an interrupted burst cycle is shown in Figure 7.16. The Intel486 Microprocessor immediately drives ADS# active to initiate a new bus cycle after RDY# is returned active. BLAST# is driven inactive one clock after ADS# begins the second bus cycle indicating that the transfer is not complete.

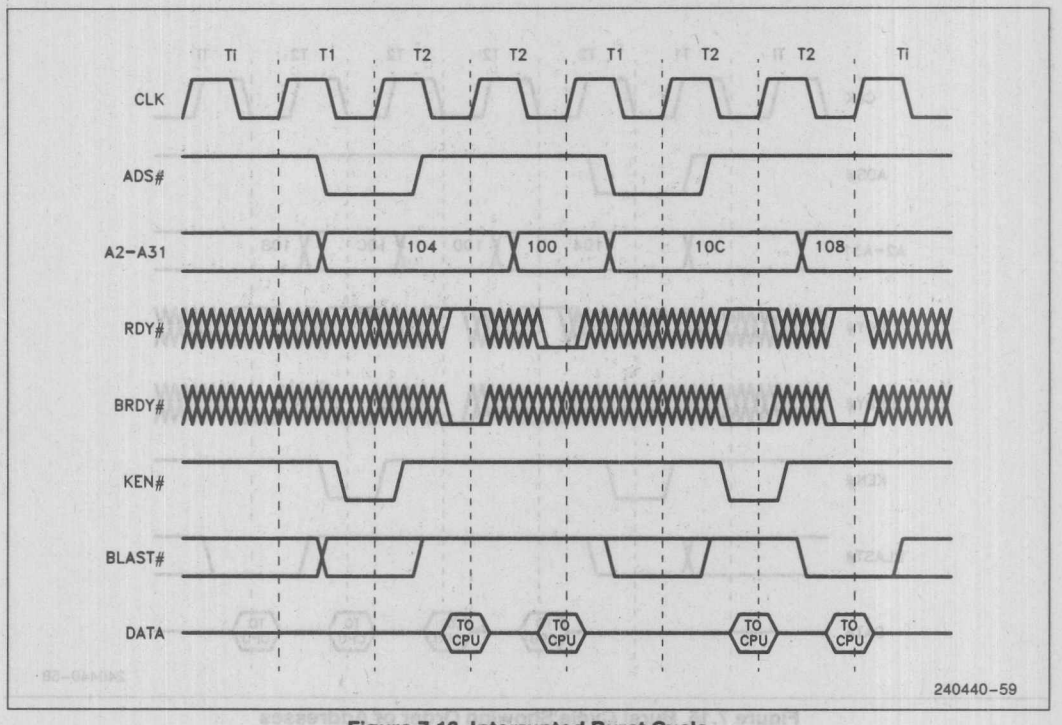


Figure 7.16. Interrupted Burst Cycle

KEN# need not be returned active in the first data cycle of the second part of the transfer in Figure 7.16. The cycle had been converted to a cache fill in the first part of the transfer and the Intel486 Microprocessor expects the cache fill to be completed. Note that the first half and second half of the transfer in Figure 7.16 are each two cycle burst transfers.

The order in which the Intel486 Microprocessor requests operands during an interrupted burst transfer is determined by Table 7.7. Mixing RDY# and BRDY# does not change the order in which operand and addresses are requested by the Intel486 Microprocessor.

An example of the order in which the Intel486 Microprocessor requests operands during a cycle in which the external system mixes RDY# and BRDY# is shown in Figure 7.17. The Intel486 Microprocessor initially requests a transfer beginning at location 104. The transfer becomes a cache line fill when the external system returns KEN# active. The first cycle of the cache fill transfers the contents of location 104 and is terminated with RDY#. The Intel486 Microprocessor drives out a new request (by asserting ADS#) to address 100. If the external system terminates the second cycle with BRDY#, the Intel486 Microprocessor will next request/expect address 10C. The correct order is determined by the first cycle in the transfer, which may not be the first cycle in the burst if the system mixes RDY# with BRDY#.

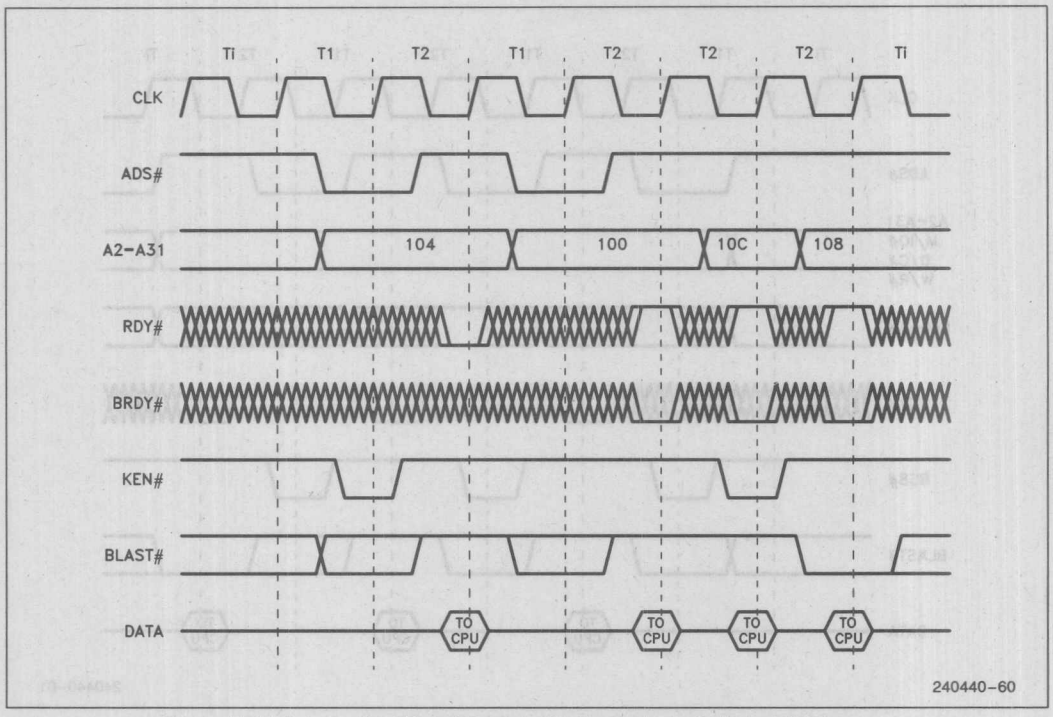


Figure 7.17. Interrupted Burst Cycle with Unobvious Order of Addresses

7.2.5 8- AND 16-BIT CYCLES

The Intel486 Microprocessor supports both 16- and 8-bit external busses through the BS16# and BS8# inputs. BS16# and BS8# allow the external system to specify, on a cycle by cycle basis, whether the addressed component can supply 8, 16 or 32 bits. BS16# and BS8# can be used in burst cycles as well as non-burst cycles. If both BS16# and BS8# are returned active for any bus cycle, the Intel486 Microprocessor will respond as if only BS8# were active.

The timing of BS16# and BS8# is the same as that of KEN#. BS16# and BS8# must be driven active before the first RDY# or BRDY# is driven active. Driving the BS16# and BS8# active can force the Intel486 Microprocessor to run additional cycles to complete what would have been only a single 32-bit cycle. BS8# and BS16# may change the state of BLAST# when they force subsequent cycles from the transfer.

Figure 7.18 shows an example in which BS8# forces the Intel486 Microprocessor to run two extra cycles to complete a transfer. The Intel486 Microprocessor issues a request for 24 bits of information. The external system drives BS8# active indicating that only eight bits of data can be supplied per cycle. The Intel486 Microprocessor issues two extra cycles to complete the transfer.

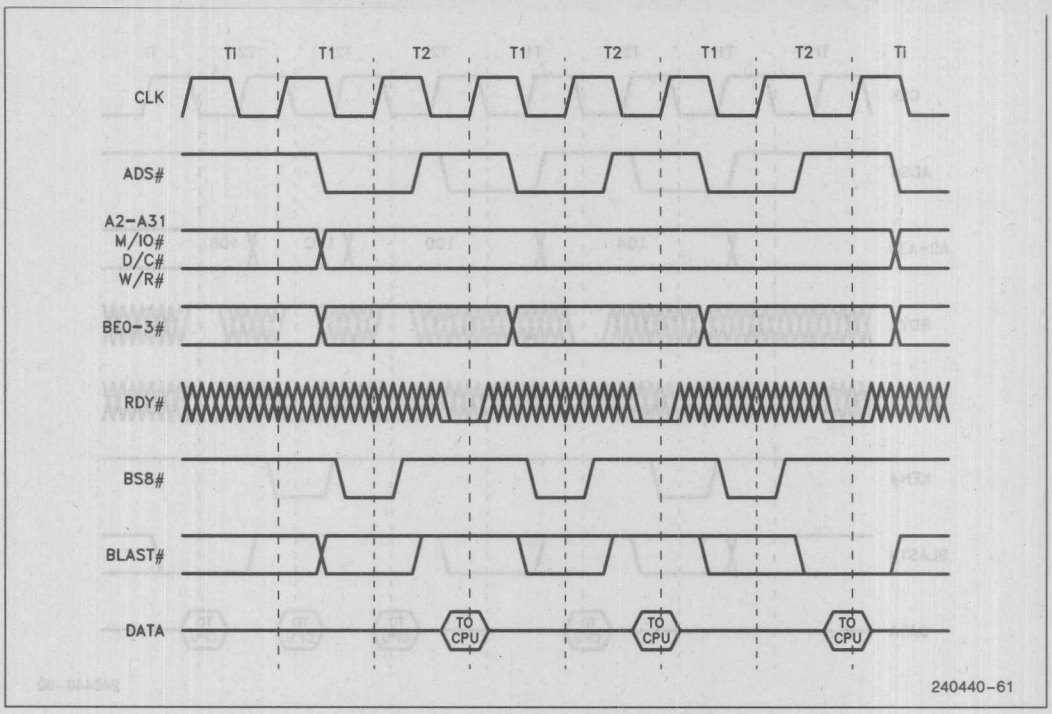


Figure 7.18. 8-Bit Bus Size Cycle

Extra cycles forced by the BS16# and BS8# should be viewed as independent bus cycles. BS16# and BS8# should be driven active for each additional cycle unless the addressed device has the ability to change the number of bytes it can return between cycles. The Intel486 Microprocessor will drive BLAST# inactive until the last cycle before the transfer is complete.

Refer to Section 7.1.3 for the sequencing of addresses while BS8# or BS16# are active.

BS8# and BS16# operate during burst cycles in exactly the same manner as non-burst cycles. For example, a single non-cacheable read could be transferred by the Intel486 Microprocessor as four 8-bit burst data cycles. Similarly, a single 32-bit write could be written as four 8-bit burst data cycles. An example of a burst write is shown in Figure 7.19. Burst writes can only occur if BS8# or BS16# is asserted.

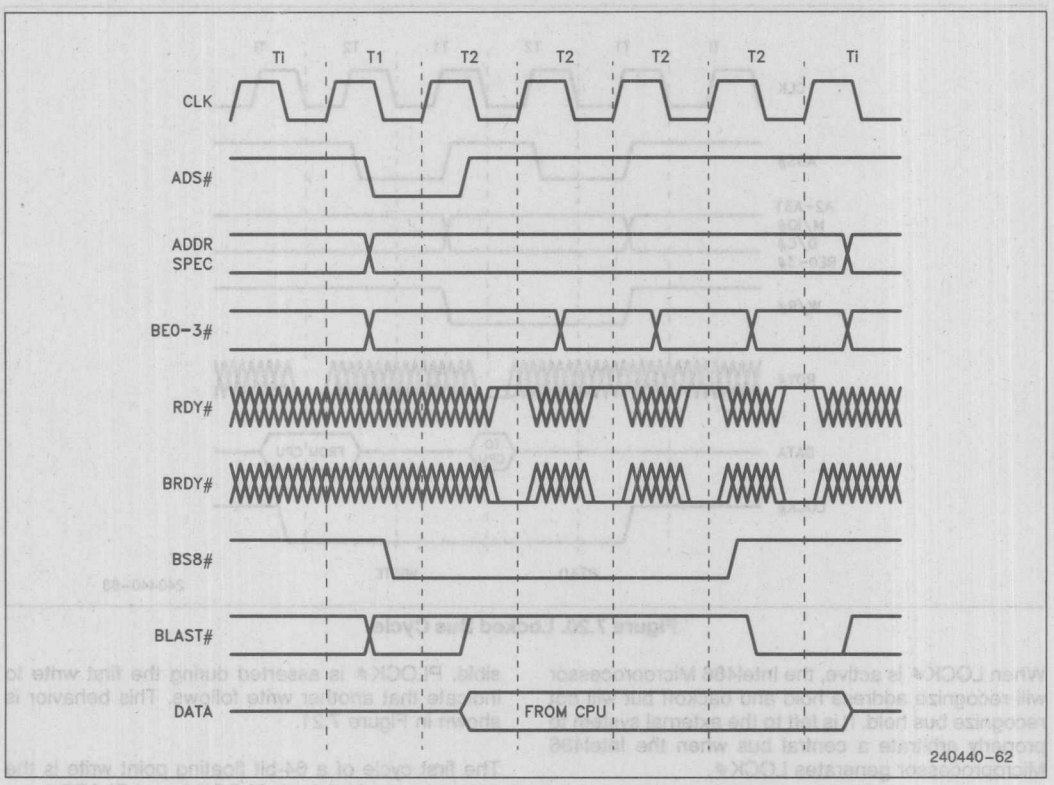


Figure 7.19. Burst Write as a Result of BS8# or BS16#

7.2.6 LOCKED CYCLES

Locked cycles are generated in software for any instruction that performs a read-modify-write operation. During a read-modify-write operation the processor can read and modify a variable in external memory and be assured that the variable is not accessed between the read and write.

Locked cycles are automatically generated during certain bus transfers. The xchg (exchange) instruction generates a locked cycle when one of its operands is memory based. Locked cycles are generated when a segment or page table entry is updated and during interrupt acknowledge cycles. Locked cycles are also generated when the LOCK instruction prefix is used with selected instructions.

Locked cycles are implemented in hardware with the LOCK# pin. When LOCK# is active, the processor is performing a read-modify-write operation and the external bus should not be relinquished until the cycle is complete. Multiple reads or writes can be locked. A locked cycle is shown in Figure 7.20. LOCK# goes active with the address and bus definition pins at the beginning of the first read cycle and remains active until RDY# is returned for the last write cycle. For unaligned 32 bits read-modify-write operation, the LOCK# remains active for the entire duration of the multiple cycle. It will go inactive when RDY# is returned for the last write cycle.

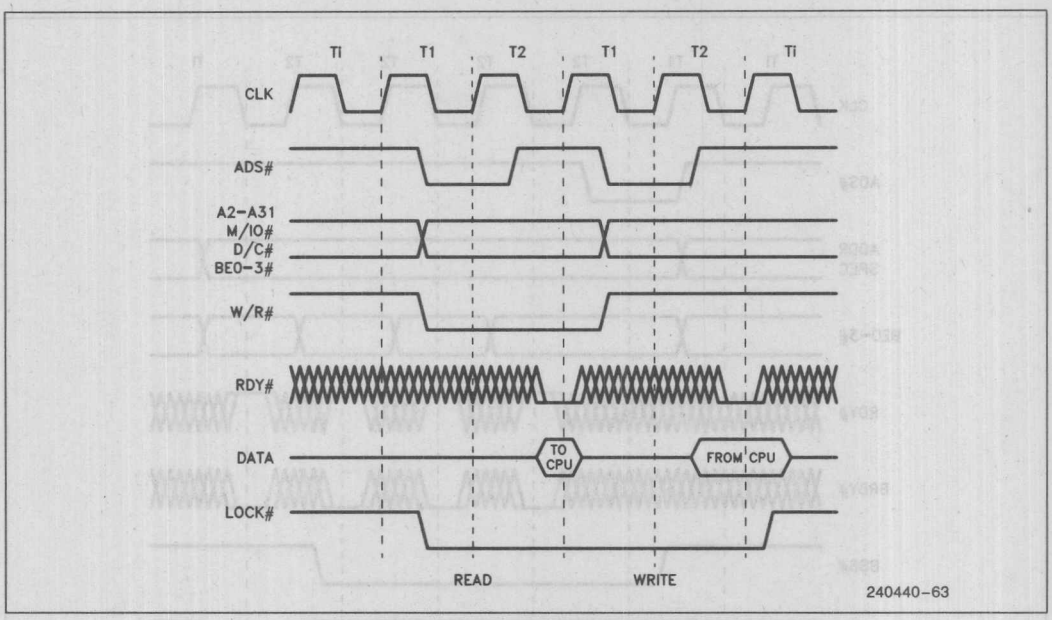


Figure 7.20. Locked Bus Cycle

When LOCK# is active, the Intel486 Microprocessor will recognize address hold and backoff but will not recognize bus hold. It is left to the external system to properly arbitrate a central bus when the Intel486 Microprocessor generates LOCK#.

7.2.7 PSEUDO-LOCKED CYCLES

Pseudo-locked cycles assure that no other master will be given control of the bus during operand transfers which take more than one bus cycle. Examples include 64-bit floating point read and writes, 64-bit descriptor loads and cache line fills.

Pseudo-locked transfers are indicated by the PLOCK# pin. The memory operands must be aligned for correct operation of a pseudo-locked cycle.

PLOCK# need not be examined during burst reads. A 64-bit aligned operand can be retrieved in one burst (note: this is only valid in systems that do not interrupt bursts).

The system must examine PLOCK# during 64-bit writes since the Intel486 Microprocessor cannot burst write more than 32 bits. However, burst can be used within each 32-bit write cycle if BS8# or BS16# is asserted. BLAST will be deasserted in response to BS8# or BS16#. A 64-bit write will be driven out as two non-burst bus cycles. BLAST# is asserted during both writes since a burst is not possible. PLOCK# is asserted during the first write to indicate that another write follows. This behavior is shown in Figure 7.21.

The first cycle of a 64-bit floating point write is the only case in which both PLOCK# and BLAST# are asserted. Normally PLOCK# and BLAST# are the inverse of each other.

During all of the cycles where PLOCK# is asserted, HOLD is not acknowledged until the cycle completes. This results in a large HOLD latency, especially when BS8# or BS16# is asserted. To reduce the HOLD latency during these cycles, windows are available between transfers to allow HOLD to be acknowledged during non-cacheable, non-bursted code prefetches. PLOCK# will be asserted since BLAST# is negated, but it is ignored and HOLD is recognized during the prefetch.

PLOCK# can change several times during a cycle settling to its final value in the clock ready is returned.

7.2.8 INVALIDATE CYCLES

Invalidate cycles are needed to keep the Intel486 Microprocessor's internal cache contents consistent with external memory. The Intel486 microprocessor contains a mechanism for listening to writes by other devices to external memory. When the processor finds a write to a Section of external memory con-

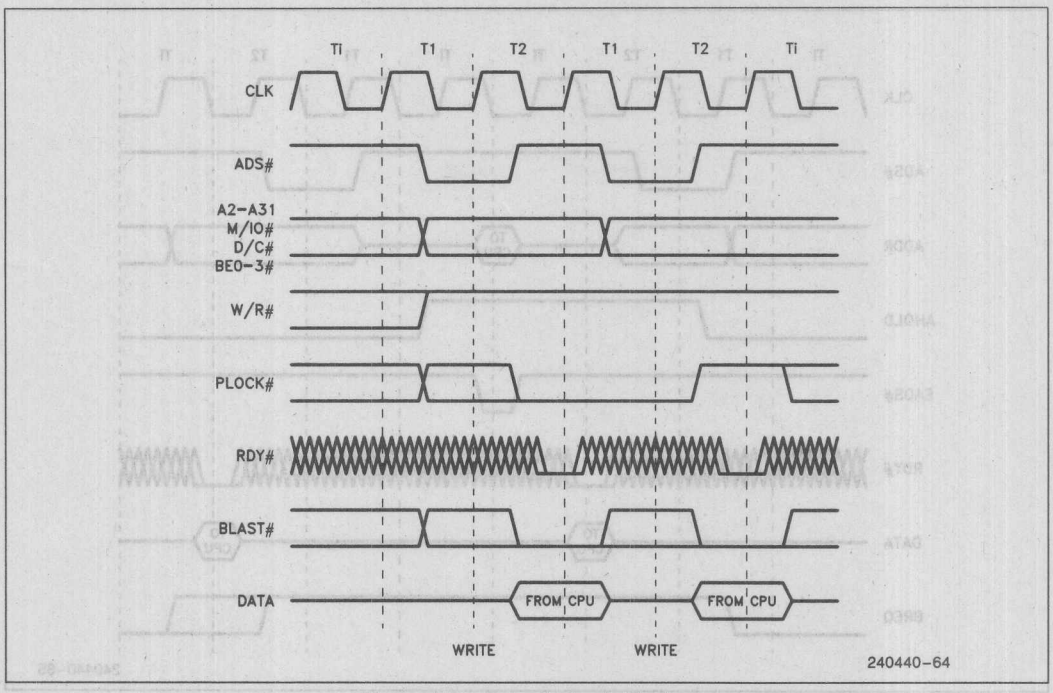


Figure 7.21. Pseudo Lock Timing

tained in its internal cache, the processor's internal copy is invalidated.

Invalidations use two pins, address hold request (AHOLD) and valid external address (EADS#). There are two steps in an invalidation cycle. First, the external system asserts the AHOLD input forcing the Intel486 Microprocessor to immediately relinquish its address bus. Next, the external system asserts EADS# indicating that a valid address is on the Intel486 Microprocessor's address bus. EADS# and the invalidation address, Figure 7-22 shows the fastest possible invalidation cycle. The Intel486 cycle CPU recognizes AHOLD on one CLK edge and floats the address bus in response. To allow the address bus to float and avoid contention, EADS# and the invalidation address should not be driven until the following CLK edge. The microprocessor reads the address over its address lines. If the microprocessor finds this address in its internal cache, the cache entry is invalidated. Note that the Intel486 Microprocessor's address bus is input/output unlike the 386 Microprocessor's bus, which is output only.

The Intel486 Microprocessor immediately relinquishes its address bus in the next clock upon assertion of AHOLD. For example, the bus could be 3 wait states into a read cycle. If AHOLD is activated, the Intel486 Microprocessor will immediately float its address bus before ready is returned terminating the bus cycle.

When AHOLD is asserted only the address bus is floated, the data bus can remain active. Data can be returned for a previously specified bus cycle during address hold (see Figures 7.22, 7.23).

EADS# is normally asserted when an external master drives an address onto the bus. AHOLD need not be driven for EADS# to generate an internal invalidate. If EADS# alone is asserted while the Intel486 Microprocessor is driving the address bus, it is possible that the invalidation address will come from the Intel486 Microprocessor itself.

Note that it is also possible to run an invalidation cycle by asserting EADS# when HOLD or BOFF# is asserted.

Running an invalidate cycle prevents the Intel486 Microprocessor cache from satisfying other internal requests, so invalidations should be run only when necessary. The fastest possible invalidate cycle is shown in Figure 7.22, while a more realistic invalidation cycle is shown in 7.23. Both of the examples take one clock of cache access from the rest of the Intel486 Microprocessor.

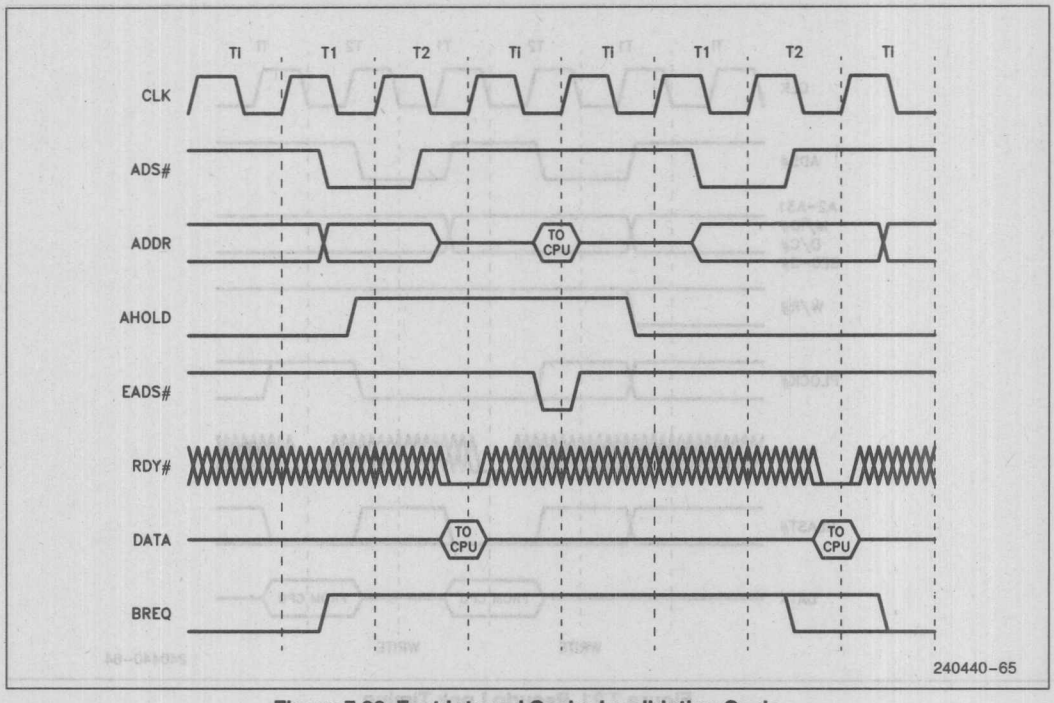


Figure 7.22. Fast Internal Cache Invalidation Cycle

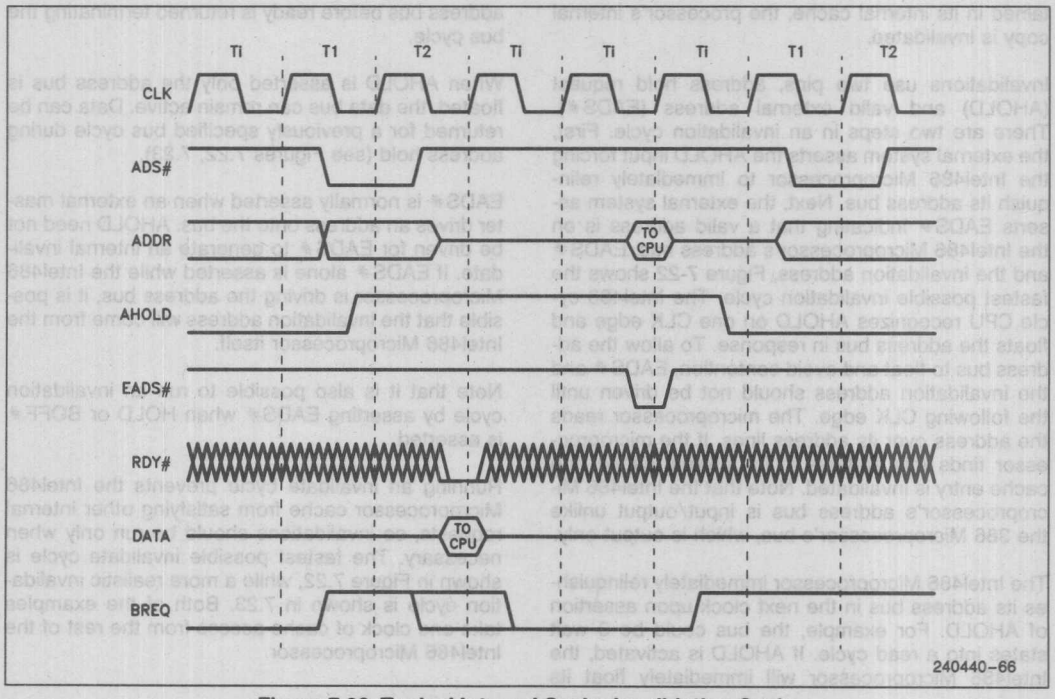


Figure 7.23. Typical Internal Cache Invalidation Cycle

7.2.8.1 Rate of Invalidate Cycles

The Intel486 Microprocessor can accept one invalidate per clock except in the last clock of a line fill. One invalidate per clock is possible as long as EADS# is negated in ONE or BOTH of the following cases:

1. In the clock RDY# or BRDY# is returned for the last time.
2. In the clock following RDY# or BRDY# being returned for the last time.

This definition allows two system designs. Simple designs can restrict invalidates to one every other clock. The simple design need not track bus activity. Alternatively, systems can request one invalidate per clock provided that the bus is monitored.

7.2.8.2 Running Invalidate Cycles Concurrently with Line Fills

Precautions are necessary to avoid caching stale data in the Intel486 Microprocessor's cache in a system with a second level cache. An example of a system with a second level cache is shown in Figure 7.24. An external device can be writing to main memory over the system bus while the Intel486 Microprocessor is retrieving data from the second level cache. The Intel486 Microprocessor will need to invalidate a line in its internal cache if the external device is writing to a main memory address also contained in the Intel486 Microprocessor's cache.

A potential problem exists if the external device is writing to an address in external memory, and at the same time the Intel486 Microprocessor is reading data from the same address in the second level cache. The system must force an invalidation cycle to invalidate the data that the Intel486 Microprocessor has requested during the line fill.

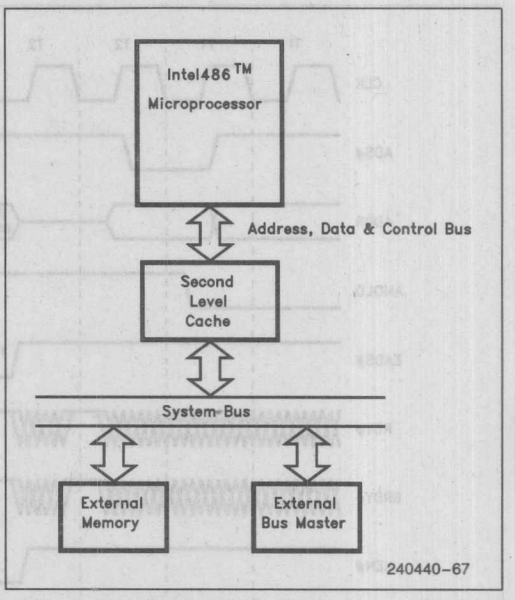
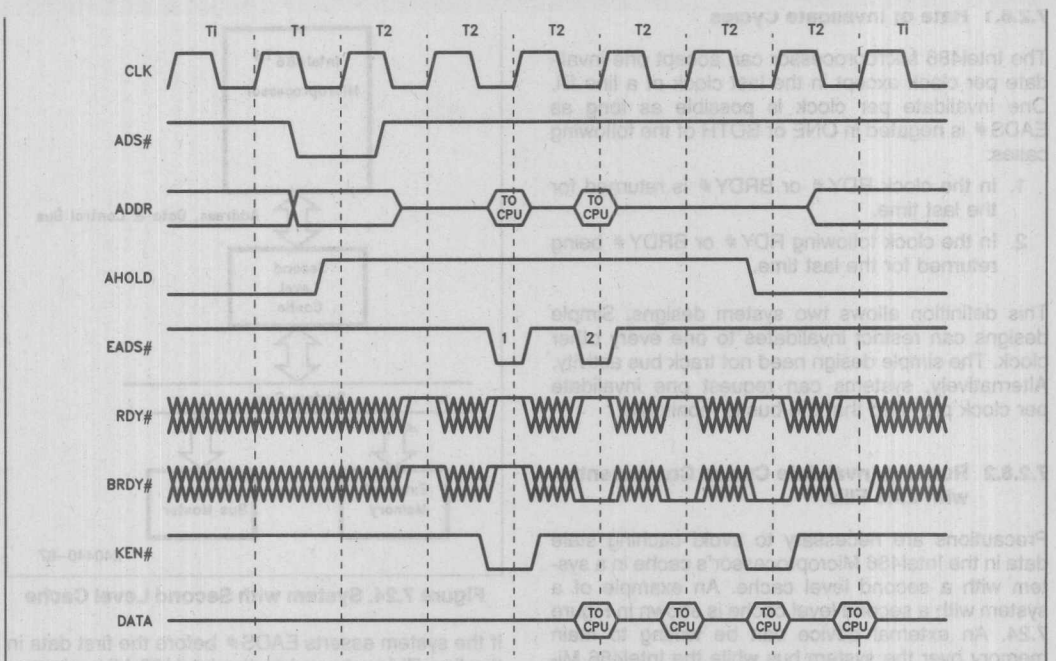


Figure 7.24. System with Second Level Cache

If the system asserts EADS# before the first data in the line fill is returned to the Intel486 Microprocessor, the system must return data consistent with the new data in the external memory upon resumption of the line fill after the invalidation cycle. This is illustrated by the asserted EADS# signal labeled 1 in Figure 7.25.

If the system asserts EADS# at the same time or after the first data in the line fill is returned (in the same clock that the first RDY# or BRDY# is returned or any subsequent clock in the line fill) the data will be read into the Intel486 Microprocessors input buffers but it will not be stored in the on-chip cache. This is illustrated by asserted EADS# signal labeled 2 in Figure 7.25. The stale data will be used to satisfy the request that initiated the cache fill cycle.



NOTES:
1. Data returned must be consistent if its address equals the invalidation address in this clock
2. Data returned will not be cached if its address equals the invalidation address in this clock

240440-68

Figure 7.25. Cache Invalidation Cycle Concurrent with Line Fill

7.2.9 BUS HOLD

The Intel486 Microprocessor provides a bus hold, hold acknowledge protocol using the bus hold request (HOLD) and bus hold acknowledge (HLDA) pins. Asserting the HOLD input indicates that another bus master desires control of the Intel486 Microprocessor's bus. The processor will respond by floating its bus and driving HLDA active when the current bus cycle, or sequence of locked cycles is complete. An example of a HOLD/HLDA transaction is shown in Figure 7.26a. Unlike the 386 Microprocessor, the Intel486 Microprocessor can respond to HOLD by floating its bus and asserting HLDA while RESET is asserted.

Note that HOLD will be recognized during un-aligned writes (less than or equal to 32-bits) with BLAST# being active for each write. For greater than 32-bit or un-aligned write, HOLD# recognition is prevented by PLOCK# getting asserted.

For cacheable and nonbursted or bursted cycles, HOLD is acknowledged during backoff only if HOLD and BOFF# are asserted during an active bus cycle (after ADS# asserted) and before the first RDY# or BRDY# has been returned (see Figure 7.26b). The order in which HOLD and BOFF# go active is unimportant (so long as both are active prior to the first RDY#/BRDY# returned by the system). Figure 7.26b shows the case where HOLD is asserted first; HOLD could be asserted simultaneously or after BOFF# and still be acknowledged.

The pins floated during bus hold are: BE0#–BE3#, PCD, PWT, W/R#, D/C#, M/IO#, LOCK#, PLOCK#, ADS#, BLAST#, D0–D31, A2–A31, DP0–DP3.

7.2.10 INTERRUPT ACKNOWLEDGE

The Intel486 Microprocessor generates interrupt acknowledge cycles in response to maskable interrupt requests generated on the interrupt request input (INTR) pin. Interrupt acknowledge cycles have a unique cycle type generated on the cycle type pins.

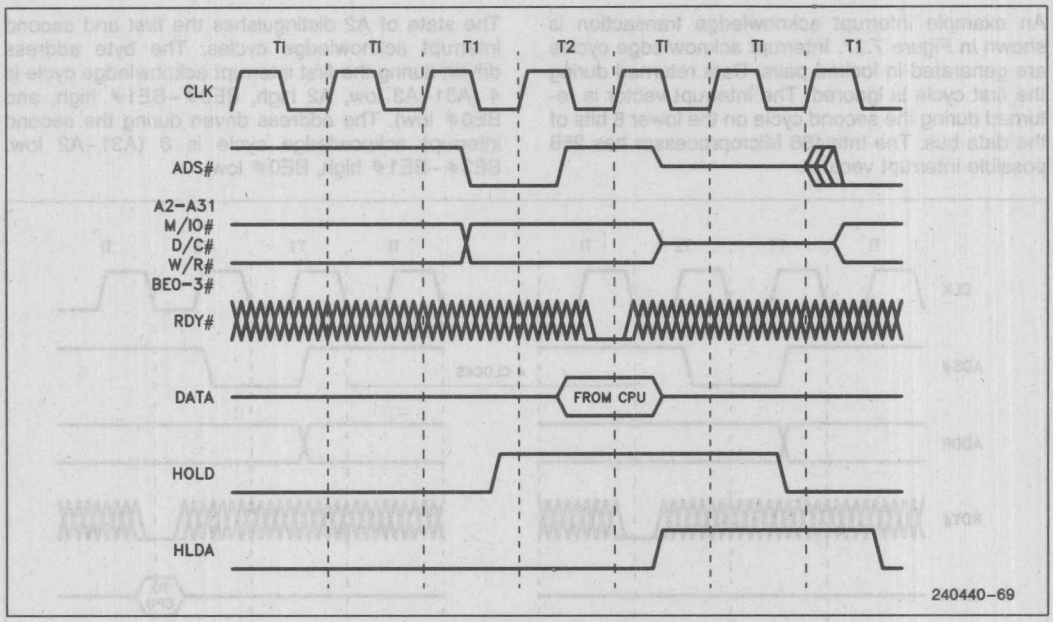


Figure 7.26a. HOLD/HLDA Cycles

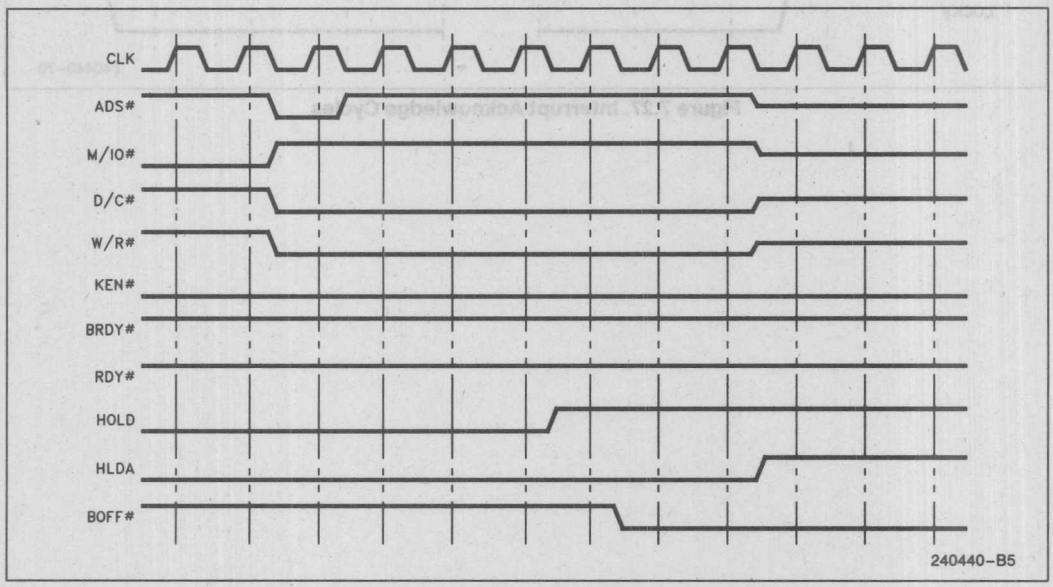


Figure 7.26b. HOLD Request Acknowledged during BOFF#

An example interrupt acknowledge transaction is shown in Figure 7.27. Interrupt acknowledge cycles are generated in locked pairs. Data returned during the first cycle is ignored. The interrupt vector is returned during the second cycle on the lower 8 bits of the data bus. The Intel486 Microprocessor has 256 possible interrupt vectors.

The state of A2 distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A31–A3 low, A2 high, BE3#–BE1# high, and BE0# low). The address driven during the second interrupt acknowledge cycle is 0 (A31–A2 low, BE3#–BE1# high, BE0# low).

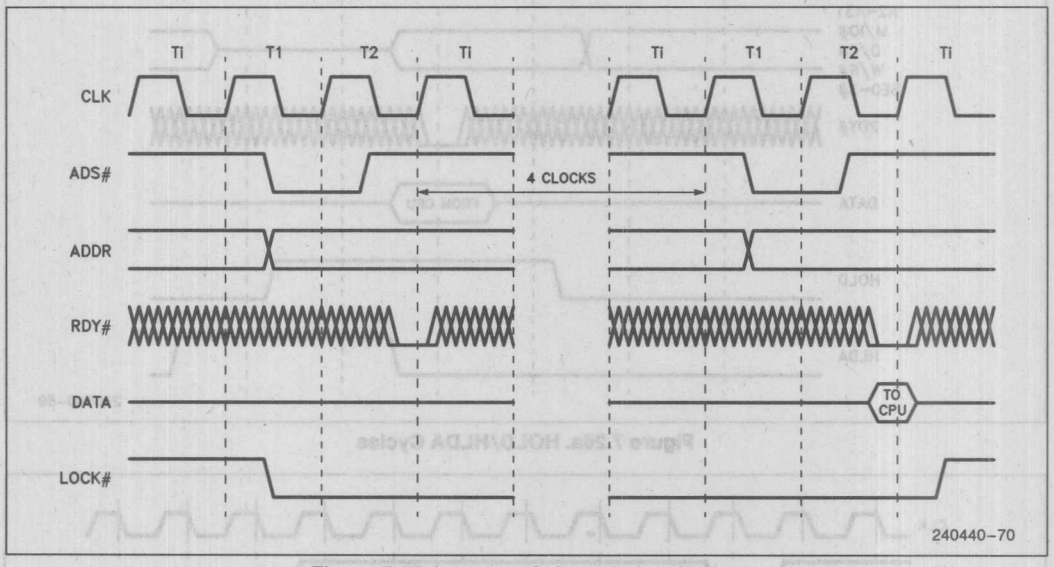


Figure 7.27. Interrupt Acknowledge Cycles

Each of the interrupt acknowledge cycles are terminated when the external system returns RDY# or BRDY#. Wait states can be added by withholding RDY# or BRDY#. The Intel486 Microprocessor automatically generates four idle clocks between the first and second cycles to allow for 8259A recovery time.

7.2.11 SPECIAL BUS CYCLES

The Intel486 Microprocessor provides four special bus cycles to indicate that certain instructions have been executed, or certain conditions have occurred internally. The special bus cycles in Table 7.9 are defined when the bus cycle definition pins are in the following state: M/IO# = 0, D/C# = 0 and W/R# = 1. During these cycles the address bus is driven low while the data bus is undefined.

Two of the special cycles indicate halt or shutdown. Another special cycle is generated when the Intel486 Microprocessor executes an INVD (invalidate data cache) instruction and could be used to flush an external cache. The Write Back cycle is generated when the Intel486 Microprocessor executes the WBINVD (write-back invalidate data cache) instruction and could be used to synchronize an external write-back cache.

The external hardware must acknowledge these special bus cycles by returning RDY# or BRDY#.

Table 7.9. Special Bus Cycle Encoding

BE3#	BE2#	BE1#	BE0#	Special Bus Cycle
1	1	1	0	Shutdown
1	1	0	1	Flush
1	0	1	1	Halt
0	1	1	1	Write Back

7.2.11.1 Halt Indication Cycle

The Intel486 Microprocessor halts as a result of executing a HALT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the bus definition signals in special bus cycle state and a byte address of 2. BE0# and BE2# are the only signals distinguishing halt indication from shutdown indication, which drives an address of 0. During the halt cycle undefined data is driven on D0–D31. The halt indication cycle must be acknowledged by RDY# or BRDY# asserted.

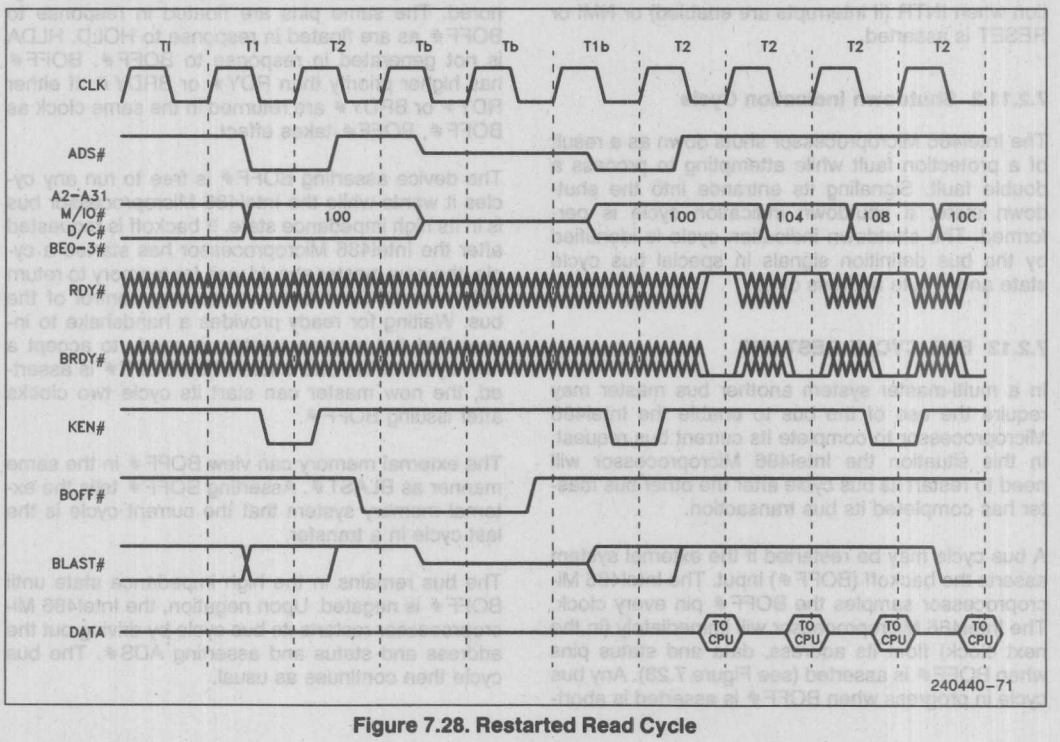


Figure 7.28. Restarted Read Cycle

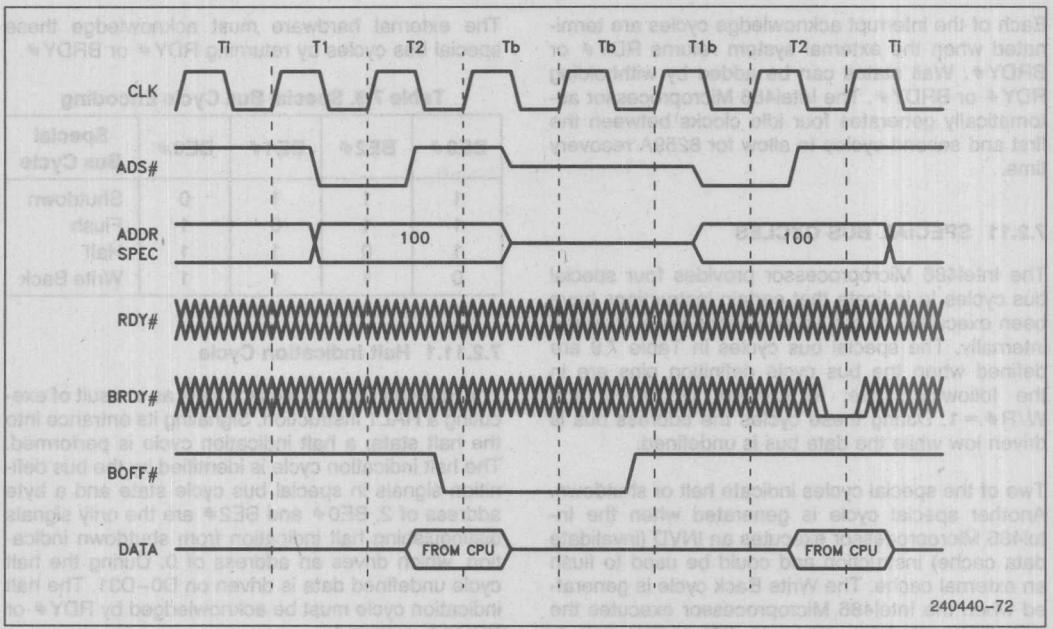


Figure 7.29. Restarted Write Cycle

A halted Intel486 Microprocessor resumes execution when INTR (if interrupts are enabled) or NMI or RESET is asserted.

7.2.11.2 Shutdown Indication Cycle

The Intel486 Microprocessor shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the bus definition signals in special bus cycle state and a byte address of 0.

7.2.12 BUS CYCLE RESTART

In a multi-master system another bus master may require the use of the bus to enable the Intel486 Microprocessor to complete its current bus request. In this situation the Intel486 Microprocessor will need to restart its bus cycle after the other bus master has completed its bus transaction.

A bus cycle may be restarted if the external system asserts the backoff (BOFF#) input. The Intel486 Microprocessor samples the BOFF# pin every clock. The Intel486 Microprocessor will immediately (in the next clock) float its address, data and status pins when BOFF# is asserted (see Figure 7.28). Any bus cycle in progress when BOFF# is asserted is aborted and any data returned to the processor is ignored. The same pins are floated in response to BOFF# as are floated in response to HOLD. HLDA is not generated in response to BOFF#. BOFF# has higher priority than RDY# or BRDY#. If either RDY# or BRDY# are returned in the same clock as BOFF#, BOFF# takes effect.

The device asserting BOFF# is free to run any cycles it wants while the Intel486 Microprocessor bus is in its high impedance state. If backoff is requested after the Intel486 Microprocessor has started a cycle, the new master should wait for memory to return RDY# or BRDY# before assuming control of the bus. Waiting for ready provides a handshake to insure that the memory system is ready to accept a new cycle. If the bus is idle when BOFF# is asserted, the new master can start its cycle two clocks after issuing BOFF#.

The external memory can view BOFF# in the same manner as BLAST#. Asserting BOFF# tells the external memory system that the current cycle is the last cycle in a transfer.

The bus remains in the high impedance state until BOFF# is negated. Upon negation, the Intel486 Microprocessor restarts its bus cycle by driving out the address and status and asserting ADS#. The bus cycle then continues as usual.

Asserting BOFF# during a burst, BS8# or BS16# cycle will force the Intel486 Microprocessor to ignore data returned for that cycle only. Data from previous cycles will still be valid. For example, if BOFF# is asserted on the third BRDY# of a burst, the Intel486 Microprocessor assumes the data returned with the first and second BRDY#'s is correct and restarts the burst beginning with the third item. The same rule applies to transfers broken into multiple cycle by BS8# or BS16#.

Asserting BOFF# in the same clock as ADS# will cause the Intel486 Microprocessor to float its bus in the next clock and leave ADS# floating low. Since ADS# is floating low, a peripheral may think that a new bus cycle has begun even-though the cycle was aborted. There are two possible solutions to this problem. The first is to have all devices recognize this condition and ignore ADS# until ready comes back. The second approach is to use a "two clock" backoff: in the first clock AHOLD is asserted, and in the second clock BOFF# is asserted. This guarantees that ADS# will not be floating low. This is only necessary in systems where BOFF# may be asserted in the same clock as ADS#.

7.2.13 BUS STATES

A bus state diagram is shown in Figure 7.30. A description of the signals used in the diagram is given in Table 7.10.

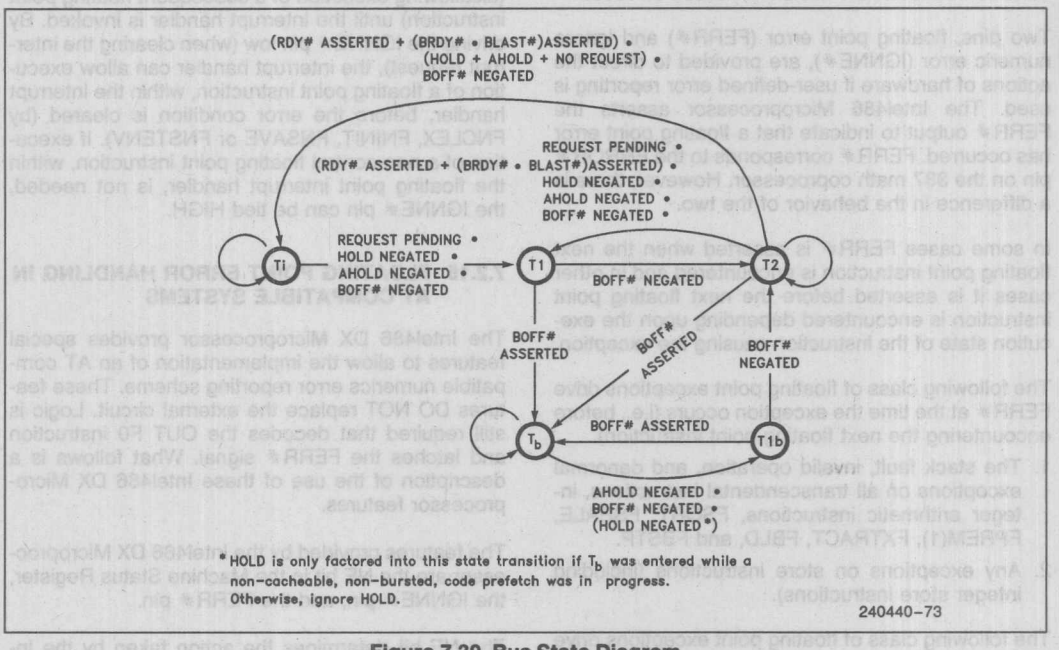


Figure 7.30. Bus State Diagram

Table 7.10. Bus State Description

State	Means
Ti	Bus is idle. Address and status signals may be driven to undefined values, or the bus may be floated to a high impedance state.
T1	First clock cycle of a bus cycle. Valid address and status are driven and ADS# is asserted.
T2	Second and subsequent clock cycles of a bus cycle. Data is driven if the cycle is a write, or data is expected if the cycle is a read. RDY# and BRDY# are sampled.
T1b	First clock cycle of a restarted bus cycle. Valid address and status are driven and ADS# is asserted.
Tb	Second and subsequent clock cycles of an aborted bus cycle.

The Intel486 Microprocessor provides two options for reporting floating point errors. The simplest method is to raise interrupt 16 whenever an unmasked floating point error occurs. This option may be enabled by setting the NE bit in control register 0 (CR0).

The Intel486 Microprocessor also provides the option of allowing external hardware to determine how floating point errors are reported. This option is necessary for compatibility with the error reporting scheme used in DOS based systems. The NE bit must be cleared in CR0 to enable user-defined error reporting. User-defined error reporting is the default condition because the NE bit is cleared on reset.

Two pins, floating point error (FERR#) and ignore numeric error (IGNNE#), are provided to direct the actions of hardware if user-defined error reporting is used. The Intel486 Microprocessor asserts the FERR# output to indicate that a floating point error has occurred. FERR# corresponds to the ERROR# pin on the 387 math coprocessor. However, there is a difference in the behavior of the two.

In some cases FERR# is asserted when the next floating point instruction is encountered and in other cases it is asserted before the next floating point instruction is encountered depending upon the execution state of the instruction causing the exception.

The following class of floating point exceptions drive FERR# at the time the exception occurs (i.e., before encountering the next floating point instruction).

1. The stack fault, invalid operation, and denormal exceptions on all transcendental instructions, integer arithmetic instructions, FSQRT, FSEALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exceptions on store instructions (including integer store instructions).

The following class of floating point exceptions drive FERR# only after encountering the next floating point instruction.

1. Exceptions other than on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exception on all basic arithmetic, load, compare, and control instructions (i.e., all other instructions).

For both sets of exceptions above, the 387 Math Coprocessor asserts ERROR# when the error occurs and does not wait for the next floating point instruction to be encountered.

IGNNE# is an input to the Intel486 Microprocessor. When the NE bit in CR0 is cleared, and IGNNE# is asserted, the Intel486 Microprocessor will ignore a user floating point error and continue executing floating point instructions. When IGNNE# is negated, the Intel486 Microprocessor will freeze on floating point instructions which get errors (except for the control instructions FNCLEX, FNINIT, FNSAVE, FNSTENV, FNSTCW, FNSTSW, FNSTSW AX, FNENI, FNDISI and FNSETPM). IGNNE# may be asynchronous to the Intel486 clock.

In systems with user-defined error reporting, the FERR# pin is connected to the interrupt controller. When an unmasked floating point error occurs, an interrupt is raised. If IGNNE# is high at the time of this interrupt, the Intel486 Microprocessor will freeze (disallowing execution of a subsequent floating point instruction) until the interrupt handler is invoked. By driving the IGNNE# pin low (when clearing the interrupt request), the interrupt handler can allow execution of a floating point instruction, within the interrupt handler, before the error condition is cleared (by FNCLEX, FNINIT, FNSAVE or FNSTENV). If execution of a non-control floating point instruction, within the floating point interrupt handler, is not needed, the IGNNE# pin can be tied HIGH.

7.2.15 FLOATING POINT ERROR HANDLING IN AT COMPATIBLE SYSTEMS

The Intel486 DX Microprocessor provides special features to allow the implementation of an AT compatible numerics error reporting scheme. These features DO NOT replace the external circuit. Logic is still required that decodes the OUT F0 instruction and latches the FERR# signal. What follows is a description of the use of these Intel486 DX Microprocessor features.

The features provided by the Intel486 DX Microprocessor are the NE bit in the Machine Status Register, the IGNNE# pin, and the FERR# pin.

The NE bit determines the action taken by the Intel486 DX Microprocessor when a numerics error is detected. When set this bit signals that non-DOS compatible error handling will be implemented. In this mode the Intel486 DX Microprocessor takes a software exception (16) if a numerics error is detected.

If the NE bit is reset the Intel486 DX Microprocessor uses the IGNNE# pin to allow an external circuit to control the time at which non-control numerics instructions are allowed to execute. Note that floating point control instructions such as FNINIT and FNSAVE can be executed during a floating point error condition regardless of the state of IGNNE#.

To process a floating point error in the DOS environment the following sequence must take place:

1. The error is detected by the Intel486 DX Microprocessor which activates the FERR# pin.
2. FERR# is latched so that it can be cleared by the OUT F0 instruction.
3. The latched FERR# signal activates an interrupt at the interrupt controller. This interrupt is usually handled on IRQ13.
4. The Interrupt Service Routine (ISR) handles the error and then clears the interrupt by executing an OUT instruction to port F0. The address F0 is decoded externally to clear the FERR# latch. The IGNNE# signal is also activated by the decoder output.
5. Usually the ISR then executes an FNINIT instruction or other control instruction before restarting the program. FNINIT clears the FERR# output.

Figure 7.31 illustrates the circuit required to perform this function. Note that this circuit has not been tested. It is included as an example of the required error handling logic.

Note that the IGNNE# input allows non-control instructions to be executed prior to the time the FERR# signal is reset by the Intel486 DX Microprocessor. This function is implemented to allow exact compatibility with the AT implementation. Most programs reinitialize the floating point unit before continuing after an error is detected. The floating point unit can be reinitialized using one of the following four instructions: FCLEX, FINIT, FSAVE, FSTENV.

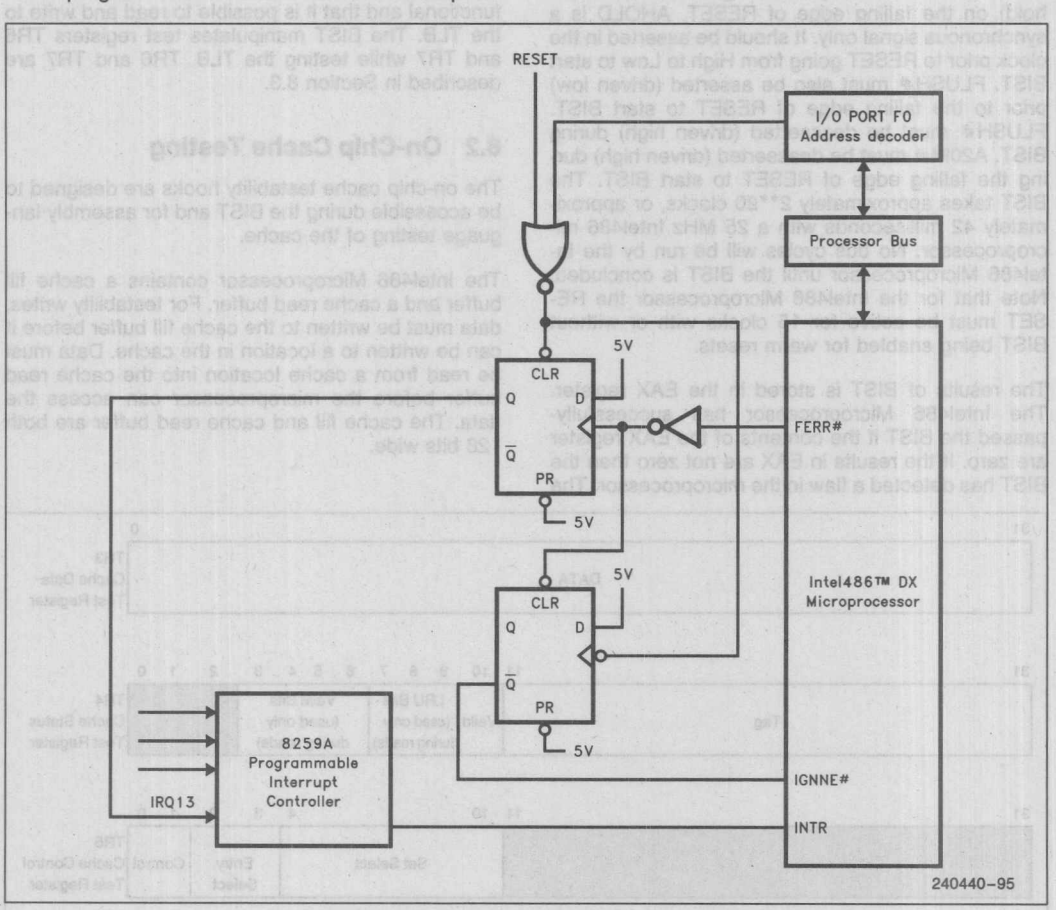


Figure 7.31. DOS Compatible Numerics Error Circuit

8.0 Intel486 CPU TESTABILITY

Testing the Intel486 Microprocessor can be divided into three categories: Built-In Self Test (BIST), Boundary Scan, and external testing. BIST performs basic device testing on the Intel486 CPU, including the non-random logic, control ROM (CROM), translation lookaside buffer (TLB), and on-chip cache memory. Boundary Scan provides additional test hooks that conform to the IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std.1149.1). The Intel486 Microprocessor also has a test mode in which all of its outputs are tristated. Additional testing can be performed by using the test registers within the Intel486 CPU.

8.1 Built-In Self Test (BIST)

The BIST is initiated by asserting AHOLD (address hold) on the falling edge of RESET. AHOLD is a synchronous signal only. It should be asserted in the clock prior to RESET going from High to Low to start BIST. FLUSH# must also be asserted (driven low) prior to the falling edge of RESET to start BIST. FLUSH# must be deasserted (driven high) during BIST. A20M# must be deasserted (driven high) during the falling edge of RESET to start BIST. The BIST takes approximately 2**20 clocks, or approximately 42 milliseconds with a 25 MHz Intel486 microprocessor. No bus cycles will be run by the Intel486 Microprocessor until the BIST is concluded. Note that for the Intel486 Microprocessor the RESET must be active for 15 clocks with or without BIST being enabled for warm resets.

The results of BIST is stored in the EAX register. The Intel486 Microprocessor has successfully-passed the BIST if the contents of the EAX register are zero. If the results in EAX are not zero then the BIST has detected a flaw in the microprocessor. The microprocessor performs reset and begins normal operation at the completion of the BIST.

The non-random logic, control ROM, on-chip cache and translation lookaside buffer (TLB) are tested during the BIST.

The cache portion of the BIST verifies that the cache is functional and that it is possible to read and write to the cache. The BIST manipulates test registers TR3, TR4 and TR5 while testing the cache. These test registers are described in Section 8.2.

The cache testing algorithm writes a value to each cache entry, reads the value back, and checks that the correct value was read back. The algorithm may be repeated more than once for each of the 512 cache entries using different constants.

The TLB portion of the BIST verifies that the TLB is functional and that it is possible to read and write to the TLB. The BIST manipulates test registers TR6 and TR7 while testing the TLB. TR6 and TR7 are described in Section 8.3.

8.2 On-Chip Cache Testing

The on-chip cache testability hooks are designed to be accessible during the BIST and for assembly language testing of the cache.

The Intel486 Microprocessor contains a cache fill buffer and a cache read buffer. For testability writes, data must be written to the cache fill buffer before it can be written to a location in the cache. Data must be read from a cache location into the cache read buffer before the microprocessor can access the data. The cache fill and cache read buffer are both 128 bits wide.

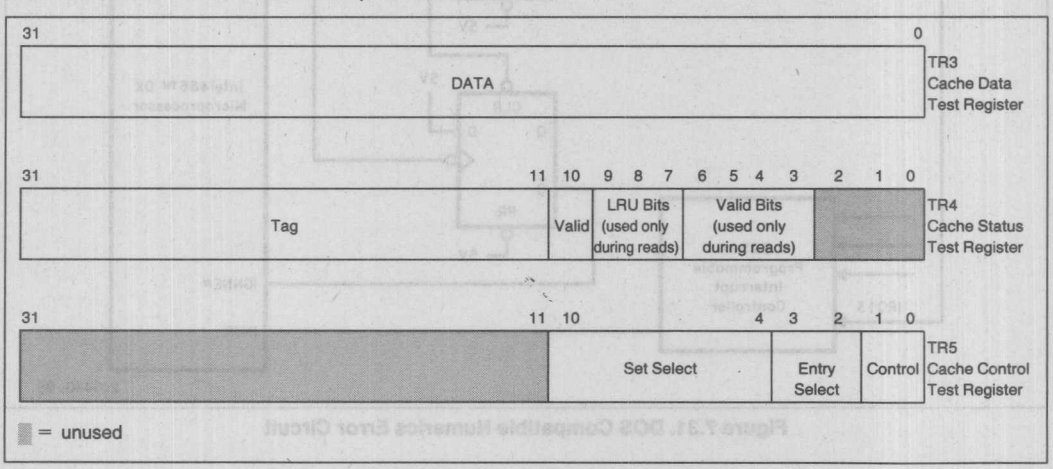


Figure 8.1. Cache Test Registers

8.2.1 CACHE TESTING REGISTERS TR3, TR4 AND TR5

Figure 8.1 shows the three cache testing registers: the Cache Data Test Register (TR3), the Cache Status Test Register (TR4) and the Cache Control Test Register (TR5). External access to these registers is provided through MOV reg,TREG and MOV TREG, reg instructions.

Cache Data Test Register: TR3

The cache fill buffer and the cache read buffer can only be accessed through TR3. Data to be written to the cache fill buffer must first be written to TR3. Data read from the cache read buffer must be loaded into TR3.

TR3 is 32 bits wide while the cache fill and read buffers are 128 bits wide. 32 bits of data must be written to TR3 four times to fill the cache fill buffer. 32 bits of data must be read from TR3 four times to empty the cache read buffer. The entry select bits in TR5 determine which 32 bits of data TR3 will access in the buffers.

Cache Status Test Register: TR4

TR4 handles tag, LRU and valid bit information during cache tests. TR4 must be loaded with a tag and a valid bit before a write to the cache. After a read from a cache entry, TR4 contains the tag and valid bit from that entry, and the LRU bits and four valid bits from the accessed set.

Cache Control Test Register: TR5

TR5 specifies which testability operation will be performed and the set and entry within the set which will be accessed.

The seven bit set select field determines which of the 128 sets will be accessed.

The functionality of the two entry select bits depend on the state of the control bits. When the fill or read buffers are being accessed, the entry select bits point to the 32-bit location in the buffer being accessed. When a cache location is specified, the entry select bits point to one of the four entries in a set. Refer to Table 8.1.

Five testability functions can be performed on the cache. The two control bits in TR5 specify the operation to be executed. The five operations are:

1. Write cache fill buffer
2. Perform a cache testability write
3. Perform a cache testability read
4. Read the cache read buffer
5. Perform a cache flush

Table 8.1 shows the encoding of the two control bits in TR5 for the cache testability functions. Table 8.1 also shows the functionality of the entry and set select bits for each control operation.

The cache tests attempt to use as much of the normal operating circuitry as possible. Therefore when cache tests are being performed, the cache must be disabled (the CD and NW bits in control register must be set to 1 to disable the cache. See Section 5).

8.2.2 CACHE TESTABILITY WRITE

A testability write to the cache is a two step process. First the cache fill buffer must be loaded with 128 bits of data and TR4 loaded with the tag and valid bit. Next the contents of the fill buffer are written to a cache location. Sample assembly code to do a write is given in Figure 8.2.

Table 8.1. Cache Control Bit Encoding and Effect of Control Bits on Entry Select and Set Select Functionality

Control Bits		Operation	Entry Select Bits Function	Set Select Bits
Bit 1	Bit 0			
0	0	Enable { Fill Buffer Write / Read Buffer Read	Select 32-bit location in fill/read buffer	—
0	1	Perform Cache Write	Select an entry in set.	Select a set to write to
1	0	Perform Cache Read	Select an entry in set.	Select a set to read from
1	1	Perform Flush Cache	—	—

Sample Assembly Code

An example assembly language sequence to perform a cache write is:

```
;
; eax. ebx. ecx. edx contain the cache line to write
; edi contains the tag information to load
; CR0 already says to enable reads/write to TR5
;
; fill the cache buffer
        mov esi,0          ; set up command
        mov tr5,esi        ; load to TR5
        mov tr3,eax        ; load data into cache fill buffer
        mov esi,4
        mov tr5,esi
        mov tr3,ebx
        mov esi,8
        mov tr5,esi
        mov tr3,ecx
        mov esi,0ch
        mov tr5,esi
        mov tr3,edx
;
; load the Cache Status Register
;
        mov tr4,edi        ; load 21-bit tag and valid bit
;
; perform the cache write
;
        mov esi,1
        mov tr5,esi        ; write the cache (set 0, entry 0)
```

An example assembly language sequence to perform a cache read is:

```
;
; data into eax, ebx, ecx, edx; status into edi
;
; read the cache line back
;
        mov esi,2
        mov tr5,esi        ; do cache testability read (set 0, entry 0)
;
; read the data from the read buffer
;
        mov esi,0
        mov tr5,esi
        mov eax,tr3
        mov esi,4
        mov tr5,esi
        mov ebx,tr3
        mov esi,8
        mov tr5,esi
        mov ecx,tr3
        mov esi,0ch
        mov tr5,esi
        mov edx,tr3
;
; read the status from TR4
;
        mov edi,tr4
```

Figure 8.2 Sample Assembly Code for Cache Testing

Loading the fill buffer is accomplished by first writing to the entry select bits in TR5 and setting the control bits in TR5 to 00. The entry select bits identify one of four 32-bit locations in the cache fill buffer to put 32 bits of data. Following the write to TR5, TR3 is written with 32 bits of data which are immediately placed in the cache fill buffer. Writing to TR3 initiates the write to the cache fill buffer. The cache fill buffer is loaded with 128 bits of data by writing to TR5 and TR3 four times using a different entry select location each time.

TR4 must be loaded with the 21-bit tag and valid bit (bit 10 in TR4) before the contents of the fill buffer are written to a cache location.

The contents of the cache fill buffer are written to a cache location by writing TR5 with a control field of 01 along with the set select and entry select fields. The set select and entry select field indicate the location in the cache to be written. The normal cache LRU update circuitry updates the internal LRU bits for the selected set.

Note that a cache testability write can only be done when the cache is disabled for replaces (the CD bit is control register 0 is reset to 1). Also note that care must be taken when directly writing to entries in the cache. If the entry is set to overlap an area of memory that is being used in external memory, that cache entry could inadvertently be used instead of the external memory. Of course, this is exactly the type of operation that one would desire if the cache were to be used as a high speed RAM.

8.2.3 CACHE TESTABILITY READ

A cache testability read is a two step process. First the contents of the cache location are read into the cache read buffer. Next the data is examined by reading it out of the read buffer. Sample assembly code to do a testability read is given in Figure 8.2.

Reading the contents of a cache location into the cache read buffer is initiated by writing TR5 with the control bits set to 10 and the desired seven-bit set select and two-bit entry select. In response to the write to TR5, TR4 is loaded with the 21-bit tag field and the single valid bit from the cache entry read. TR4 is also loaded with the three LRU bits and four valid bits corresponding to the cache set that was accessed. The cache read buffer is filled with the 128-bit value which was found in the data array at the specified location.

The contents of the read buffer are examined by performing four reads of TR3. Before reading TR3 the entry select bits in TR5 must loaded to indicate which of the four 32-bit words in the read buffer to transfer into TR3 and the control bits in TR5 must be loaded with 00. The register read of TR3 will initiate the transfer of the 32-bit value from the read buffer to the specified general purpose register.

Note that it is very important that the entire 128-bit quantity from the read buffer and also the information from TR4 be read before any memory references are allowed to occur. If memory operations are allowed to happen, the contents of the read buffer will be corrupted. This is because the testability operations use hardware that is used in normal memory accesses for the Intel486 microprocessor whether the cache is enabled or not.

8.2.4 FLUSH CACHE

The control bits in TR5 must be written with 11 to flush the cache. None of the other bits in TR5 have any meaning when 11 is written to the control bits. Flushing the cache will reset the LRU bits and the valid bits to 0, but will not change the cache tag or data arrays.

When the cache is flushed by writing to TR5 the special bus cycle indicating a cache flush to the external system is not run (see Section 7.2.11, Special Bus Cycles). The cache should be flushed with the instruction INVD (Invalidate Data Cache) instruction or the WBINVD (Write-back and Invalidate Data Cache) instruction.

8.3 Translation Lookaside Buffer (TLB) Testing

The Intel486 Microprocessor TLB testability hooks are similar to those in the 386 Microprocessor. The testability hooks have been enhanced to provide added test features and to include new features in the Intel486 Microprocessor. The TLB testability hooks are designed to be accessible during the BIST and for assembly language testing of the TLB.

8.3.1 TRANSLATION LOOKASIDE BUFFER ORGANIZATION

The Intel486 Microprocessors TLB is 4-way set associative and has space for 32 entries. The TLB is logically split into three blocks shown in Figure 8.3.

The data block is physically split into four arrays, each with space for eight entries. An entry in the data block is 22 bits wide containing a 20-bit physical address and two bits for the page attributes. The page attributes are the PCD (page cache disable) bit and the PWT (page write-through) bit. Refer to Section 4.5.4 for a discussion of the PCD and PWT bits.

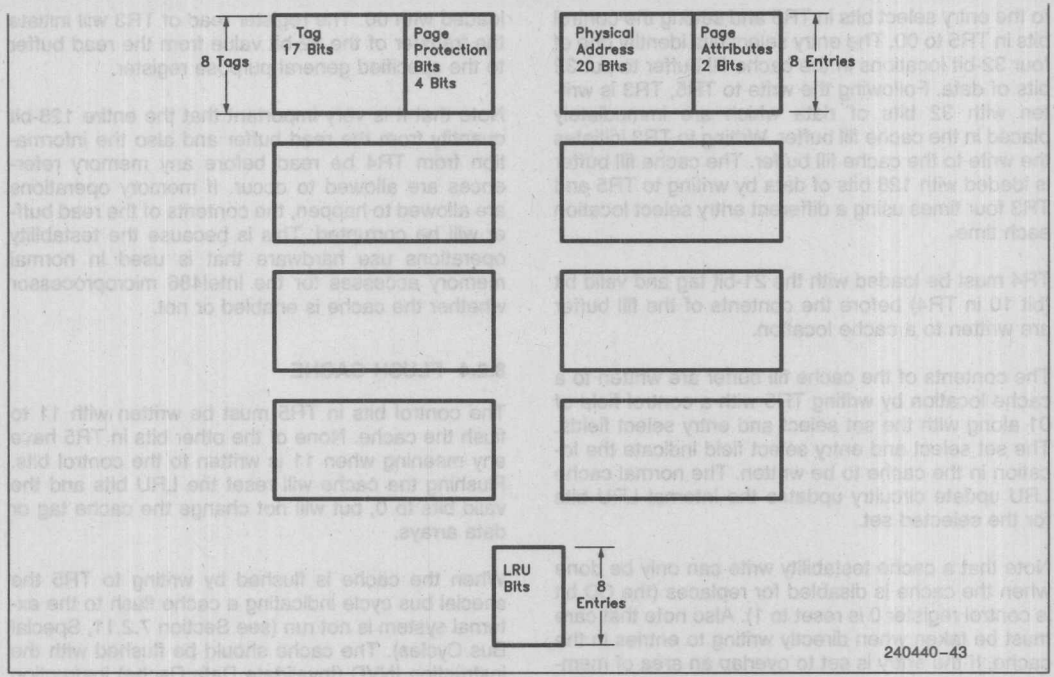


Figure 8.3. TLB Organization

The tag block is also split into four arrays, one for each of the data arrays. A tag entry is 21 bits wide containing a 17-bit linear address and four protection bits. The protection bits are valid (V), user/supervisor (U/S), read/write (R/W) and dirty (D).

The third block contains eight three bit quantities used in the pseudo least recently used (LRU) replacement algorithm. These bits are called the LRU bits. The LRU replacement algorithm used in the TLB is the same as used by the on-chip cache. For a description of this algorithm refer to Section 5.5.

8.3.2 TLB TEST REGISTERS TR6 AND TR7

The two TLB test registers are shown in Figure 8.4. TR6 is the command test register and TR7 is the data test register. External access to these registers is provided through MOV reg,TREG and MOV TREG,reg instructions.

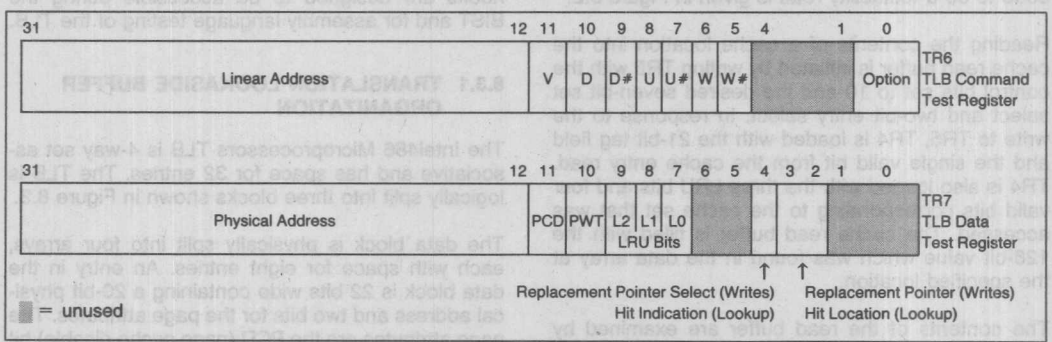


Figure 8.4. TLB Test Registers

PRELIMINARY

Command Test Register: TR6

TR6 contains the tag information and control information used in a TLB test. Loading TR6 with tag and control information initiates a TLB write or lookup test.

TR6 contains three bit fields, a 20-bit linear address (bits 12–31), seven bits for the TLB tag protection bits (bits 5-11) and one bit (bit 0) to define the type of operation to be performed on the TLB.

The 20-bit linear address forms the tag information used in the TLB access. The lower three bits of the linear address select which of the eight sets are accessed. The upper 17 bits of the linear address form the tag stored in the tag array.

The seven TLB tag protection bits are described below.

V: The valid bit for this TLB entry

D,D#: The dirty bit for/from the TLB entry

U,U#: The user/supervisor bit for/from the TLB entry

W,W#: The read/write bit for/from the TLB entry

Two bits are used to represent the D, U/S and R/W bits in the TLB tag to permit the option of a forced miss or hit during a TLB lookup operation. The forced miss or hit will occur regardless of the state of the actual bit in the TLB. The meaning of these pairs of bits is given in Table 8.2.

The operation bit in TR6 determines if the TLB test operation will be a write or a lookup. The function of the operation bit is given in Table 8.3.

Table 8.3. TR6 Operation Bit Encoding

TR6 Bit 0	TLB Operation to Be Performed
0	TLB Write
1	TLB Lookup

Data Test Register: TR7

TR7 contains the information stored or read from the data block during a TLB test operation. Before a TLB test write, TR7 contains the physical address and the page attribute bits to be stored in the entry. After a TLB test lookup hit, TR7 contains the physical address, page attributes, LRU bits and entry location from the access.

TR7 contains a 20-bit physical address (bits 12–31), two bits for PCD (bit 11) and PWT (bit 10) and three bits for the LRU bits (bits 7–9). The LRU bits in TR7 are only used during a TLB lookup test. The functionality of TR7 bit 4 differs for TLB writes and lookups. The encoding of bit 4 is defined in Tables 8.4 and 8.5. Finally TR7 contains two bits (bits 2–3) to specify a TLB replacement pointer or the location of a TLB hit.

Table 8.4. Encoding of Bit 4 of TR7 on Writes

TR7 Bit 4	Replacement Pointer Used on TLB Write
0	Pseudo-LRU Replacement Pointer
1	Data Test Register Bits 3:2

Table 8.5. Encoding of Bit 4 of TR7 on Lookups

TR7 Bit 4	Meaning after TLB Lookup Operation
0	TLB Lookup Resulted in a Miss
1	TLB Lookup Resulted in a Hit

A replacement pointer is used during a TLB write. The pointer indicates which of the four entries in an accessed set is to be written. The replacement pointer can be specified to be the internal LRU bits or bits 2–3 in TR7. The source of the replacement pointer is specified by TR7 bit 4. The encoding of bit 4 during a write is given by Table 8.4.

Note that both testability writes and lookups affect the state of the internal LRU bits regardless of the replacement pointer used. All TLB write operations (testability or normal operation) cause the written entry to become the most recently used. For example, during a testability write with the replacement pointer specified by TR7 bits 2–3, the indicated entry is written and that entry becomes the most recently used as specified by the internal LRU bits.

Table 8.2. Meaning of a Pair of TR6 Protection Bits

TR6 Protection Bit (B)	TR6 Protection Bit # (B#)	Meaning on TLB Write Operation	Meaning on TLB Lookup Operation
0	0	Undefined	Miss any TLB TAG Bit B
0	1	Write 0 to TLB TAG Bit B	Match TLB TAG Bit B if 0
1	0	Write 1 to TLB TAG Bit B	Match TLB TAG Bit B if 1
1	1	Undefined	Match any TLB TAG Bit B

There are two TLB testing operations: write entries into the TLB, and perform TLB lookups. One major enhancement over TLB testing in the 386 Microprocessor is that paging need not be disabled while executing testability writes or lookups.

Note that any time one TLB set contains the same linear address in more than one of its entries, looking up that linear address will give unpredictable results. Therefore a single linear address should not be written to one TLB set more than once.

8.3.3 TLB WRITE TEST

To perform a TLB write TR7 must be loaded followed by a TR6 load. The register operations must be performed in this order since the TLB operation is triggered by the write to TR6.

TR7 is loaded with a 20-bit physical address and values for PCD and PWT to be written to the data portion of the TLB. In addition, bit 4 of TR7 must be loaded to indicate whether to use TR7 bits 3-2 or the internal LRU bits as the replacement pointer on the TLB write operation. Note that the LRU bits in TR7 are not used in a write test.

TR6 must be written to initiate the TLB write operation. Bit 0 in TR6 must be reset to zero to indicate a TLB write. The 20-bit linear address and the seven page protection bits must also be written in TR6 to specify the tag portion of the TLB entry. Note that the three least significant bits of the linear address specify which of the eight sets in the data block will be loaded with the physical address data. Thus only 17 of the linear address bits are stored in the tag array.

8.3.4 TLB LOOKUP TEST

To perform a TLB lookup it is only necessary to write the proper tags and control information into TR6. Bit 0 in TR6 must be set to 1 to indicate a TLB lookup. TR6 must be loaded with a 20-bit linear address and the seven protection bits. To force misses and matches of the individual protection bits on TLB lookups, set the seven protection bits as specified in Table 8.2.

A TLB lookup operation is initiated by the write to TR6. TR7 will indicate the result of the lookup operation following the write to TR6. The hit/miss indication can be found in TR7 bit 4 (see Table 8.5).

TR7 will contain the following information if bit 4 indicated that the lookup test resulted in a hit. Bits 2–3 will indicate in which set the match occurred. The 22 most significant bits in TR7 will contain the physical address and page attributes contained in the entry. Bits 9–7 will contain the LRU bits associated with the accessed set. The state of the LRU bits is previous to their being updated for the current lookup.

If bit 4 in TR7 indicated that the lookup test resulted in a miss the remaining bits in TR7 are undefined.

Again it should be noted that a TLB testability lookup operation affects the state of the LRU bits. The LRU bits will be updated if a hit occurred. The entry which was hit will become the most recently used.

8.4 Tristate Output Test Mode

The Intel486 Microprocessor provides the ability to float all its outputs and bidirectional pins. This includes all pins floated during bus hold as well as pins which are never floated in normal operation of the chip (HLDA, BREQ, FERR# and PCHK#). When the Intel486 microprocessor is in the tri-state output test mode external testing can be used to test board connections.

The tri-state test mode is invoked by driving FLUSH# low for 2 clocks before and 2 clocks after RESET going low. The outputs are guaranteed to tristate no later than 10 clocks after RESET goes low (see Figure 6.4). The Intel486 Microprocessor remains in the tristate test mode until the next RESET.

8.5 Intel486™ Microprocessor Boundary Scan (JTAG)

The Intel486 Microprocessor (50 MHz version only) provides additional testability features compatible with the IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std.1149.1). The test logic provided allows for testing to insure that components function correctly, that interconnections between various components are correct, and that various components interact correctly on the printed circuit board.

The boundary scan test logic consists of a boundary scan register and support logic that are accessed through a test access port (TAP). The TAP provides a simple serial interface that makes it possible to test all signal traces with only a few probes.

The TAP can be controlled via a bus master. The bus master can be either automatic test equipment or a component (PLD) that interfaces to the four-pin test bus.

8.5.1 BOUNDARY SCAN ARCHITECTURE

The boundary scan test logic contains the following elements:

— Test access port (TAP), consisting of input pins TMS, TCK, and TDI; and output pin TDO.

— TAP controller, which interprets the inputs on the test mode select (TMS) line and performs the corresponding operation. The operations performed by the TAP include controlling the instruction and data registers within the component.

— Instruction register (IR), which accepts instruction codes shifted into the test logic on the test data input (TDI) pin. The instruction codes are used to select the specific test operation to be performed or the test data register to be accessed.

— Test data registers: The Intel486 Microprocessor contains three test data registers: Bypass register (BPR), Device Identification register (DID), and Boundary Scan register (BSR).

The instruction and test data registers are separate shift-register paths connected in parallel and have a common serial data input and a common serial data output connected to the TAP signals, TDI and TDO, respectively.

8.5.2 DATA REGISTERS

The Intel486 CPU contains the two required test data registers; bypass register and boundary scan register. In addition, they also have a device identification register.

Each test data register is serially connected to TDI and TDO, with TDI connected to the most significant bit and TDO connected to the least significant bit of the test data register. Data is shifted one stage (bit position within the register) on each rising edge of the test clock (TCK).

In addition the Intel486 CPU contains a runbist register to support the RUNBIST boundary scan instruction.

8.5.2.1 Bypass Register

The Bypass Register is a one-bit shift register that provides the minimal length path between TDI and TDO. This path can be selected when no test operation is being performed by the component to allow rapid movement of test data to and from other components on the board. While the bypass register is selected, data is transferred from TDI to TDO without inversion.

8.5.2.2 Boundary Scan Register

The Boundary Scan Register is a single shift register path containing the boundary scan cells that are connected to all input and output pins of the Intel486 CPU. Figure 8.5 shows the logical structure of the boundary scan register. While output cells determine the value of the signal driven on the corresponding pin, input cells only capture data; they do not affect the normal operation of the device. Data is transferred without inversion from TDI to TDO through the boundary scan register during scanning. The boundary scan register can be operated by the EXTEST and SAMPLE instructions. The boundary scan register order is described in Section 8.5.5.

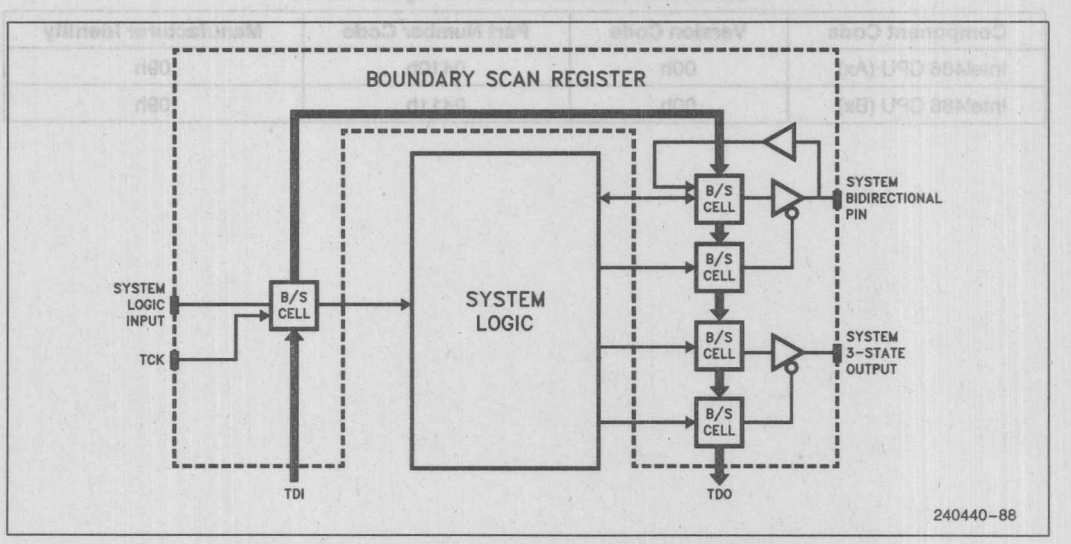


Figure 8.5. Logical Structure of Boundary Scan Register

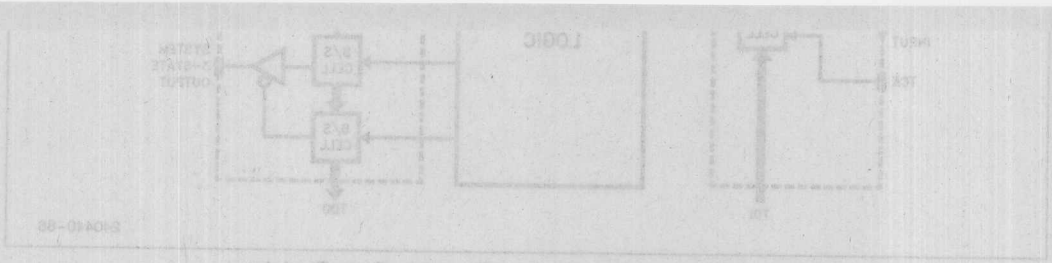


Table 8.7. Boundary Scan Instruction Codes

Instruction Code	Instruction Name
0000	EXTEST
0001	SAMPLE
0010	IDCODE
0011	PRIVATE
0100	PRIVATE
0101	PRIVATE
0110	PRIVATE
0111	PRIVATE
1000	RUNBIST
1001	PRIVATE
1010	PRIVATE
1011	PRIVATE
1100	PRIVATE
1101	PRIVATE
1110	PRIVATE
1111	BYPASS

EXTEST The instruction code is "0000". The EXTEST instruction allows testing of circuitry external to the component package, typically board interconnects. It does so by driving the values loaded into the Intel486 CPU's boundary scan register out on the output pins corresponding to each boundary scan cell and capturing the values on Intel486 CPU input pins to be loaded into their corresponding boundary scan register locations. I/O pins are selected as input or output, depending on the value loaded into their control setting locations in the boundary scan register. Values shifted into input latches in the boundary scan register are never used by the internal logic of the Intel486 CPU.

NOTE:

After using the EXTEST instruction, the Intel486 CPU must be reset before normal (non-boundary scan) use.

SAMPLE/PRELOAD The instruction code is "0001". The SAMPLE/PRELOAD has two functions that it performs. When the TAP controller is in the Capture-DR state, the SAMPLE/PRELOAD instruction allows a "snap-shot" of the normal operation of the component without interfering with that normal operation. The instruction causes boundary scan register cells associated with outputs to sample the value being driven by the Intel486 CPU. It causes the cells associated with inputs to sample the value being driven into the Intel486 CPU. On both outputs and inputs the sampling occurs on the rising edge of TCK. When the TAP controller is in the Update-DR state, the SAMPLE/PRELOAD instruction preloads data to the device pins to be driven to the board by executing the EXTEST instruction. Data is preloaded to the pins from the boundary scan register on the falling edge of TCK.

IDCODE The instruction code is "0010". The IDCODE instruction selects the device identification register to be connected to TDI and TDO, allowing the device identification code to be shifted out of the device on TDO. Note that the device identification register is not altered by data being shifted in on TDI.

BYPASS The instruction code is "1111". The BYPASS instruction selects the bypass register to be connected to TDI or TDO, effectively bypassing the test logic on the Intel486 microprocessor by reducing the shift length of the device to one bit. Note than an open circuit fault in the board level test data path will cause the bypass register to be selected following an instruction scan cycle due to the pull-up resistor on the TDI input. This has been done to prevent any unwanted interference with the proper operation of the system logic.

RUNBIST The instruction code is "1000". The RUNBIST instruction selects the one (1) bit runbist register, loads a value of "1" into the runbist register, and connects it to TDO. It also initiates the built-in self test (BIST) feature of the Intel486 CPU, which is able to detect approximately 60% of the stuck-at faults on the Intel486 CPU. The Intel486 CPU AC/DC Specifications for V_{CC} and CLK must be met and reset must have been asserted at least once prior to executing the RUNBIST boundary scan instruction. After loading the RUNBIST instruction code in the instruction register, the TAP controller must be placed in the Run-Test/Idle state. BIST begins on the first rising edge of TCK after entering the Run-Test/Idle state. The TAP con-

troller must remain in the Run-Test/Idle state until BIST is completed. It requires 1.2 million clock (CLK) cycles to complete BIST and report the result to the runbist register. After completing the 1.2 million clock (CLK) cycles, the value in the runbist register should be shifted out on TDO during the Shift-DR state. A value of "0" being shifted out on TDO indicates BIST successfully completed. A value of "1" indicates a failure occurred. After executing the RUNBIST instruction, the Intel486 CPU must be reset prior to normal operation.

8.5.4 TEST ACCESS PORT (TAP) CONTROLLER

The TAP controller is a synchronous, finite state machine. It controls the sequence of operations of the test logic. The TAP controller changes state only in response to the following events:

1. a rising edge of TCK
2. power-up.

The value of the test mode state (TMS) input signal at a rising edge of TCK controls the sequence of the state changes. The state diagram for the TAP controller is shown in Figure 8.7. Test designers must consider the operation of the state machine in order to design the correct sequence of values to drive on TMS.

8.5.4.1 Test-Logic-Reset State

In this state, the test logic is disabled so that normal operation of the device can continue unhindered. This is achieved by initializing the instruction register such that the IDCODE instruction is loaded. No matter what the original state of the controller, the controller enters Test-Logic-Reset state when the TMS input is held high (1) for at least five rising edges of TCK. The controller remains in this state while TMS is high. The TAP controller is also forced to enter this state at power-up.

8.5.4.2 Run-Test/Idle State

A controller state between scan operations. Once in this state, the controller remains in this state as long

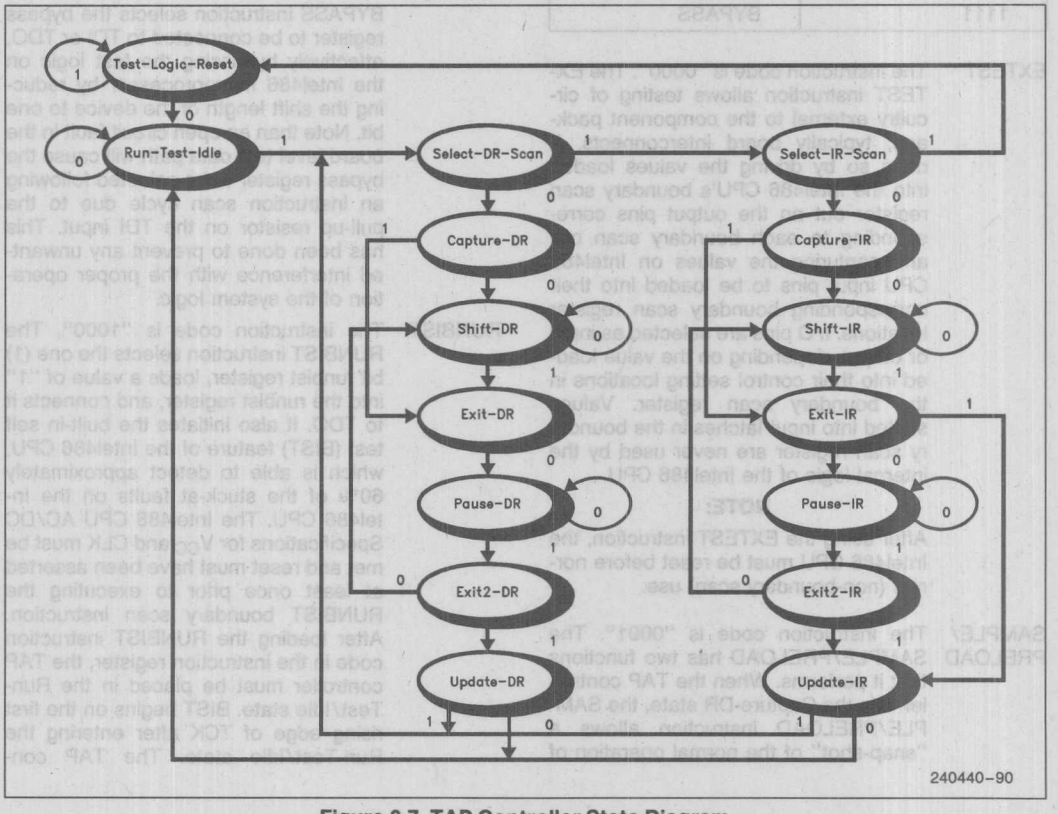


Figure 8.7. TAP Controller State Diagram

as TMS is held low. In devices supporting the RUNBIST instruction, the BIST is performed during this state and the result is reported in the runbist register. For instruction not causing functions to execute during this state, no activity occurs in the test logic. The instruction register and all test data registers retain their previous state. When TMS is high and a rising edge is applied to TCK, the controller moves to the Select-DR state.

8.5.4.3 Select-DR-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held low and a rising edge is applied to TCK when in this state, the controller moves into the Capture-DR state, and a scan sequence for the selected test data register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves to the Select-IR-Scan state.

The instruction does not change in this state.

8.5.4.4 Capture-DR State

In this state, the boundary scan register captures input pin data if the current instruction is EXTEST or SAMPLE/PRELOAD. The other test data registers, which do not have parallel input, are not changed.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-DR state if TMS is high or the Shift-DR state if TMS is low.

8.5.4.5 Shift-DR State

In this controller state, the test data register connected between TDI and TDO as a result of the current instruction, shifts data one stage toward its serial output on each rising edge of TCK.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-DR state if TMS is high or remains in the Shift-DR state if TMS is low.

8.5.4.6 Exit1-DR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-DR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.7 Pause-Dr State

The pause state allows the test controller to temporarily halt the shifting of data through the test data register in the serial path between TDI and TDO. An example of using this state could be to allow a tester to reload its pin memory from disk during application of a long test sequence.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves to the Exit2-DR state.

8.5.4.8 Exit2-DR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Shift-DR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.9 Update-DR State

The boundary scan register is provided with a latched parallel output to prevent changes at the parallel output while data is shifted in response to the EXTEST and SAMPLE/PRELOAD instructions. When the TAP controller is in this state and the boundary scan register is selected, data is latched onto the parallel output of this register from the shift-register path on the falling edge of TCK. The data held at the latched parallel output does not change other than in this state.

All shift-register stages in test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.10 Select-IR-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held low and a rising edge is applied to TCK when in this state, the controller moves into the Capture-IR state, and a scan sequence for the instruction register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves to the Test-Logic-Reset state.

The instruction does not change in this state.

8.5.4.11 Capture-IR State

In this controller state the shift register contained in the instruction register loads the fixed value "0001" on the rising edge of TCK.

The test data register selected by the current instruction retains it previous value during this state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-IR state if TMS is held high, or the Shift-IR state if TMS is held low.

8.5.4.12 Shift-IR State

In this state the shift register contained in the instruction register is connected between TDI and TDO and shifts data one stage towards its serial output on each rising edge of TCK.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-IR state if TMS is held high, or remains in the Shift-IR state if TMS is held low.

8.5.4.13 Exit1-IR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-IR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.14 Pause-IR State

The pause state allows the test controller to temporarily halt the shifting of data through the instruction register.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves to the Exit2-IR state.

8.5.4.15 Exit2-IR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Shift-IR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.16 Update-IR State

The instruction shifted into the instruction register is latched onto the parallel output from the shift-register path on the falling edge of TCK. Once the new instruction has been latched, it becomes the current instruction.

Test data registers selected by the current instruction retain the previous value.

8.5.5 BOUNDARY SCAN REGISTER CELL

The boundary scan register contains a cell for each pin, as well as cells for control of I/O and tristate pins.

The following is the bit order of the Intel486 CPU boundary scan register: (from left to right and top to bottom).

TDI → WRCTL ABUSCTL BUSCTL MISCCTL ADS# BLAST# PLOCK# LOCK# PCHK# BRDY# BOFF# BS16# BS8# RDY# KEN# HOLD AHOLD CLK HLDA WR# BREQ BE0# BE1# BE2# BE3# MIO# DC# PWT PCD EADS# A20M# RESET FLUSH# INTR NMI FERR# IGNNE# D31 D30 D29 D28 D27 D26 D25 D24 DP3 D23 D22 D21 D20 D19 D18 D17 D16 DP2 D15 D14 D13 D12 D11 D10 D9 D8 DP1 D7 D6 D5 D4 D3 D2 D1 D0 DP0 A31 A30 A29 A28 A27 A26 A25 A24 A23 A22 A21 A20 A19 A18 A17 A16 A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 RESERVED A5 A4 A3 A2 → TDO

"RESERVED" corresponds to no connect "NC" signals on the Intel486 CPU.

All the *CTL cells are control cells that are used to select the direction of bidirectional pins or tristate output pins. If "1" is loaded into the control cell (*CTL), the associated pin(s) are tristated or selected as input. The following lists the control cells and their corresponding pins.

1. WRCTL controls the D31–0 and DP3–0 pins.
2. ABUSCTL controls the A31–A2 pins.
3. BUSCTL controls the ADS#, BLAST#, PLOCK#, LOCK#, WR#, BE0#, BE1#, BE2#, BE3#, MIO#, DC#, PWT, and PCD pins.
4. MISCCTL controls the PCHK#, HLDA, BREQ, and FERR# pins.

8.5.6 TAP CONTROLLER INITIALIZATION

The TAP controller is automatically initialized when a device is powered up. In addition, the TAP controller can be initialized by applying a high signal level on the TMS input for five TCK periods.

8.5.7 BOUNDARY SCAN DESCRIPTION LANGUAGE (BSDL)

Available through Intel.

9.0 DEBUGGING SUPPORT

The Intel486 Microprocessor provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

1) the code execution breakpoint opcode (0CCH),
2) the single-step capability provided by the TF bit in the flag register, and
3) the code and data breakpoint capability provided by the Debug Registers DR0–3, DR6, and DR7.

9.1 Breakpoint Instruction

A single-byte-opcode breakpoint instruction is available for use by software debuggers. The breakpoint opcode is 0CCH, and generates an exception 3 trap when executed. In typical use, a debugger program can "plant" the breakpoint instruction at all desired code execution breakpoints. The single-byte breakpoint opcode is an alias for the two-byte general software interrupt instruction, INT n, where n=3. The only difference between INT 3 (0CCh) and INT n is that INT 3 is never IOPL-sensitive but INT n is IOPL-sensitive in Protected Mode and Virtual 8086 Mode.

9.2 Single-Step Trap

If the single-step flag (TF, bit 8) in the EFLAG register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto vectored to exception number 1. Precisely, exception 1 occurs as a trap after the instruction following the instruction which set TF. In typical practice, a debugger sets the TF bit of a flag register image on the debugger's stack. It then typically transfers control to the user program and loads the flag image with a signal instruction, the IRET instruction. The single-step trap occurs after executing one instruction of the user program.

Since the exception 1 occurs as a trap (that is, it occurs after the instruction has already executed), the CS:EIP pushed onto the debugger's stack points to the next unexecuted instruction of the program being debugged. An exception 1 handler, merely by ending with an IRET instruction, can therefore efficiently support single-stepping through a user program.

9.3 Debug Registers

The Debug Registers are an advanced debugging feature of the Intel486 Microprocessor. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT3 breakpoint opcode.

The Intel486 Microprocessor contains six Debug Registers, providing the ability to specify up to four distinct breakpoints addresses, breakpoint control options, and read breakpoint status. Initially after reset, breakpoints are in the disabled state. Therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are autovectored to exception number 1.

9.3.1 LINEAR ADDRESS BREAKPOINT REGISTERS (DR0–DR3)

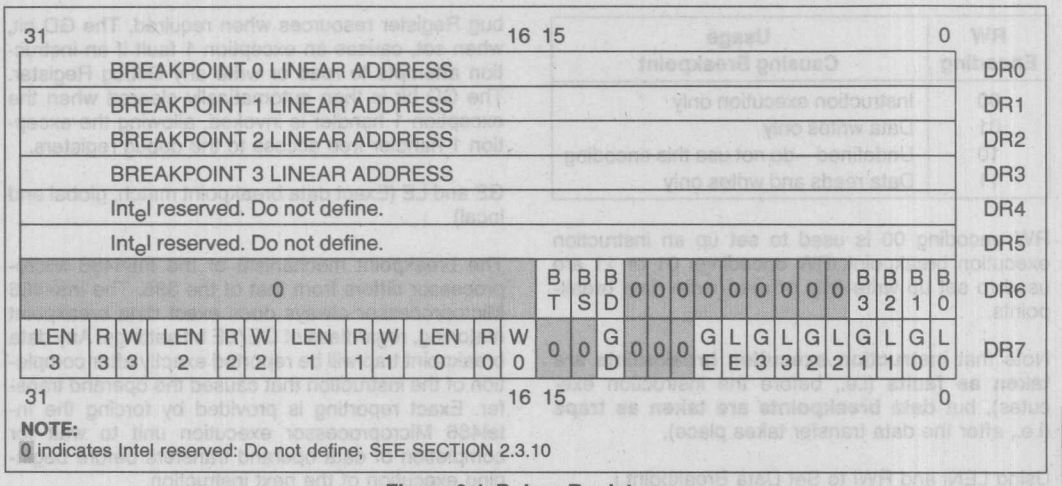
Up to four breakpoint addresses can be specified by writing into Debug Registers DR0–DR3, shown in Figure 9.1. The breakpoint addresses specified are 32-bit linear addresses. Intel486 Microprocessor hardware continuously compares the linear breakpoint addresses in DR0–DR3 with the linear addresses generated by executing software (a linear address is the result of computing the effective address and adding the 32-bit segment base address). Note that if paging is not enabled the linear address equals the physical address. If paging is enabled, the linear address is translated to a physical 32-bit address by the on-chip paging unit. Regardless of whether paging is enabled or not, however, the breakpoint registers hold linear addresses.

9.3.2 DEBUG CONTROL REGISTER (DR7)

A Debug Control Register, DR7 shown in Figure 9.1, allows several debug control functions such as enabling the breakpoints and setting up other control options for the breakpoints. The fields within the Debug Control Register, DR7, are as follows:

LENi (breakpoint length specification bits)

A 2-bit LEN field exists for each of the four breakpoints. LEN specifies the length of the associated breakpoint field. The choices for data breakpoints are: 1 byte, 2 bytes, and 4 bytes. Instruction execution breakpoints must have a length of 1 (LENi = 00). Encoding of the LENi field is as follows:



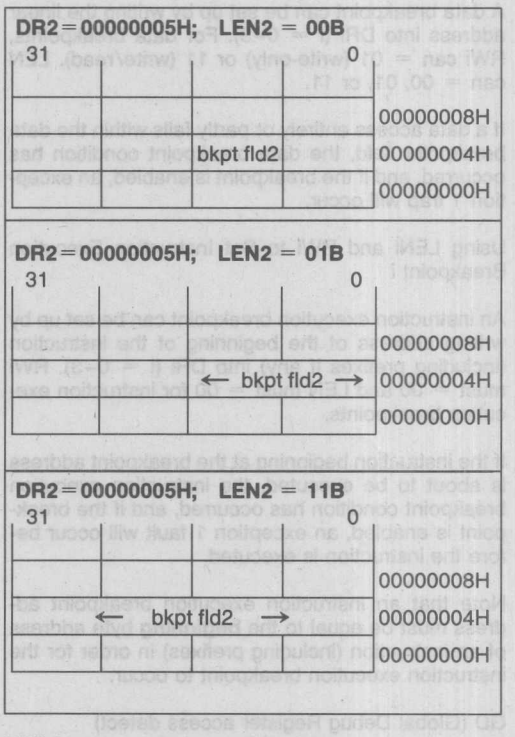
NOTE:
0 indicates Intel reserved: Do not define; SEE SECTION 2.3.10

Figure 9.1. Debug Registers

LENi Encoding	Breakpoint Field Width	Usage of Least Significant Bits in Breakpoint Address Register i, (i=0–3)
00	1 byte	All 32-bits used to specify a single-byte breakpoint field.
01	2 bytes	A1–A31 used to specify a two-byte, word-aligned breakpoint field. A0 in Breakpoint Address Register is not used.
10	Undefined—do not use this encoding	
11	4 bytes	A2–A31 used to specify a four-byte, dword-aligned breakpoint field. A0 and A1 in Breakpoint Address Register are not used.

The LENi field controls the size of breakpoint field i by controlling whether all low-order linear address bits in the breakpoint address register are used to detect the breakpoint event. Therefore, all breakpoint fields are aligned; 2-byte breakpoint fields begin on Word boundaries, and 4-byte breakpoint fields begin on Dword boundaries.

The following is an example of various size breakpoint fields. Assume the breakpoint linear address in DR2 is 00000005H. In that situation, the following illustration indicates the region of the breakpoint field for lengths of 1, 2, or 4 bytes.



RWi (memory access qualifier bits)

A 2-bit RW field exists for each of the four breakpoints. The 2-bit RW field specifies the type of usage which must occur in order to activate the associated breakpoint.

RW Encoding	Usage Causing Breakpoint
00	Instruction execution only
01	Data writes only
10	Undefined—do not use this encoding
11	Data reads and writes only

RW encoding 00 is used to set up an instruction execution breakpoint. RW encodings 01 or 11 are used to set up write-only or read/write data breakpoints.

Note that **instruction execution breakpoints are taken as faults** (i.e., before the instruction executes), but **data breakpoints are taken as traps** (i.e., after the data transfer takes place).

Using LENi and RWi to Set Data Breakpoint i

A data breakpoint can be set up by writing the linear address into DRi (i = 0–3). For data breakpoints, RWi can = 01 (write-only) or 11 (write/read). LEN can = 00, 01, or 11.

If a data access entirely or partly falls within the data breakpoint field, the data breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 trap will occur.

Using LENi and RWi to Set Instruction Execution Breakpoint i

An instruction execution breakpoint can be set up by writing address of the beginning of the instruction (including prefixes if any) into DRi (i = 0–3). RWi must = 00 and LEN must = 00 for instruction execution breakpoints.

If the instruction beginning at the breakpoint address is about to be executed, the instruction execution breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 fault will occur before the instruction is executed.

Note that an instruction execution breakpoint address must be equal to the **beginning** byte address of an instruction (including prefixes) in order for the instruction execution breakpoint to occur.

GD (Global Debug Register access detect)

The Debug Registers can only be accessed in Real Mode or at privilege level 0 in Protected Mode. The GD bit, when set, provides extra protection against **any** Debug Register access even in Real Mode or at privilege level 0 in Protected Mode. This additional protection feature is provided to guarantee that a software debugger can have full control over the Debug Register resources when required. The GD bit, when set, causes an exception 1 fault if an instruction attempts to read or write any Debug Register. The GD bit is then automatically cleared when the exception 1 handler is invoked, allowing the exception 1 handler free access to the debug registers.

GE and LE (Exact data breakpoint match, global and local)

The breakpoint mechanism of the Intel486 Microprocessor differs from that of the 386. The Intel486 Microprocessor always does exact data breakpoint matching, regardless of GE/LE bit settings. Any data breakpoint trap will be reported exactly after completion of the instruction that caused the operand transfer. Exact reporting is provided by forcing the Intel486 Microprocessor execution unit to wait for completion of data operand transfers before beginning execution of the next instruction.

When the Intel486 Microprocessor performs a task switch, the LE bit is cleared. Thus, the LE bit supports fast task switching out of tasks, that have enabled the exact data breakpoint match for their task-local breakpoints. The LE bit is cleared by the processor during a task switch, to avoid having exact data breakpoint match enabled in the new task. Note that exact data breakpoint match must be re-enabled under software control.

The Intel486 Microprocessor GE bit is unaffected during a task switch. The GE bit supports exact data breakpoint match that is to remain enabled during all tasks executing in the system.

Note that **instruction execution** breakpoints are always reported exactly.

Gi and Li (breakpoint enable, global and local)

If either Gi or Li is set then the associated breakpoint (as defined by the linear address in DRi, the length in LENi and the usage criteria in RWi) is enabled. If either Gi or Li is set, and the Intel486 Microprocessor detects the ith breakpoint condition, then the exception 1 handler is invoked.

When the Intel486 Microprocessor performs a task switch to a new Task State Segment (TSS), all Li bits are cleared. Thus, the Li bits support fast task switching out of tasks that use some task-local breakpoint registers. The Li bits are cleared by the processor during a task switch, to avoid spurious exceptions in the new task. Note that the breakpoints must be re-enabled under software control.

All Intel486 Microprocessor Gi bits are unaffected during a task switch. The Gi bits support breakpoints that are active in all tasks executing in the system.

9.3.3 DEBUG STATUS REGISTER (DR6)

A Debug Status Register, DR6 shown in Figure 9.1, allows the exception 1 handler to easily determine why it was invoked. Note the exception 1 handler can be invoked as a result of one of several events:

1) DR0 Breakpoint fault/trap.
2) DR1 Breakpoint fault/trap.
3) DR2 Breakpoint fault/trap.
4) DR3 Breakpoint fault/trap.
5) Single-step (TF) trap.
6) Task switch trap.
7) Fault due to attempted debug register access when GD = 1.

The Debug Status Register contains single-bit flags for each of the possible events invoking exception 1. Note below that some of these events are faults (exception taken before the instruction is executed), while other events are traps (exception taken after the debug events occurred).

The flags in DR6 are set by the hardware but never cleared by hardware. Exception 1 handler software should clear DR6 before returning to the user program to avoid future confusion in identifying the source of exception 1.

The fields within the Debug Status Register, DR6, are as follows:

Bi (debug fault/trap due to breakpoint 0–3)

Four breakpoint indicator flags, B0–B3, correspond one-to-one with the breakpoint registers in DR0–DR3. A flag Bi is set when the condition described by DRi, LENi, and RWi occurs.

If Gi or Li is set, and if the ith breakpoint is detected, the processor will invoke the exception 1 handler. The exception is handled as a fault if an instruction execution breakpoint occurred, or as a trap if a data breakpoint occurred.

IMPORTANT NOTE: A flag Bi is set whenever the hardware detects a match condition on **enabled** breakpoint i. Whenever a match is detected on at least one **enabled** breakpoint i, the hardware immediately sets all Bi bits corresponding to breakpoint conditions matching at that instant, whether enabled **or not.** Therefore, the exception 1 handler may see that multiple Bi bits are set, but only set Bi bits corresponding to **enabled** breakpoints (Li or Gi set) are **true** indications of why the exception 1 handler was invoked.

BD (debug fault due to attempted register access when GD bit set)

This bit is set if the exception 1 handler was invoked due to an instruction attempting to read or write to the debug registers when GD bit was set. If such an event occurs, then the GD bit is automatically cleared when the exception 1 handler is invoked, allowing handler access to the debug registers.

BS (debug trap due to single-step)

This bit is set if the exception 1 handler was invoked due to the TF bit in the flag register being set (for single-stepping).

BT (debug trap due to task switch)

This bit is set if the exception 1 handler was invoked due to a task switch occurring to a task having a Intel486 Microprocessor TSS with the T bit set. Note the task switch into the new task occurs normally, but before the first instruction of the task is executed, the exception 1 handler is invoked. With respect to the task switch operation, the operation is considered to be a trap.

9.3.4 USE OF RESUME FLAG (RF) IN FLAG REGISTER

The Resume Flag (RF) in the flag word can suppress an instruction execution breakpoint when the exception 1 handler returns to a user program at a user address which is also an instruction execution breakpoint.

10.0 INSTRUCTION SET SUMMARY

This section describes the Intel486 Microprocessor instruction set. Tables 10.1 through 10.3 list all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in Section 10.2, which completely describes the encoding structure and the definition of all fields occurring within the Intel486 Microprocessor instructions.

10.1 Intel486™ Microprocessor Instruction Encoding and Clock Count Summary

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Tables 10.1 through 10.3 by the processor clock period (e.g., 40 ns for a 25 MHz Intel486 Microprocessor).

For more detailed information on the encodings of instructions, refer to Section 10.2 Instruction Encodings. Section 10.2 explains the general structure of instruction encodings, and defines exactly the encodings of all fields contained within the instruction.

INSTRUCTION CLOCK COUNT ASSUMPTIONS

The Intel486 Microprocessor instruction clock count tables give clock counts assuming data and instruction accesses hit in the cache. A separate penalty column defines clocks to add if a data access misses in the cache. The combined instruction and data cache hit rate is over 90%.

A cache miss will force the Intel486 Microprocessor to run an external bus cycle. The Intel486 Microprocessor 32-bit burst bus is defined as r−b−w.

Where:

r = The number of clocks in the first cycle of a burst read or the number of clocks per data cycle in a non-burst read.

b = The number of clocks for the second and subsequent cycles in a burst read.

w = The number of clocks for a write.

The fastest bus the Intel486 microprocessor can support is 2−1−2 assuming 0 wait states. The clock counts in the cache miss penalty column assume a 2−1−2 bus. For slower busses add r−2 clocks to the cache miss penalty for the first dword accessed. Other factors also affect instruction clock counts.

Instruction Clock Count Assumptions

1. The external bus is available for reads or writes at all times. Else add clocks to reads until the bus is available.
2. Accesses are aligned. Add three clocks to each misaligned access.
3. Cache fills complete before subsequent accesses to the same line. If a read misses the cache during a cache fill due to a previous read or prefetch, the read must wait for the cache fill to complete. If a read or write accesses a cache line still being filled, it must wait for the fill to complete.
4. If an effective address is calculated, the base register is not the destination register of the preceding instruction. If the base register is the destination register of the preceding instruction add 1 to the clock counts shown. Back-to-back PUSH and POP instructions are not affected by this rule.
5. An effective address calculation uses one base register and does not use an index register. However, if the effective address calculation uses an index register, 1 clock **may** be added to the clock count shown.
6. The target of a jump is in the cache. If not, add r clocks for accessing the destination instruction of a jump. If the destination instruction is not completely contained in the first dword read, add a maximum of 3b clocks. If the destination instruction is not completely contained in the first 16 byte burst, add a maximum of another r+3b clocks.
7. If no write buffer delay, w clocks are added only in the case in which all write buffers are full. Typically, this case rarely occurs.
8. Displacement and immediate not used together. If displacement and immediate used together, 1 clock **may** be added to the clock count shown.
9. No invalidate cycles. Add a delay of 1 clock for each invalidate cycle if the invalidate cycle contends for the internal cache/external bus when the Intel486 CPU needs to use it.
10. Page translation hits in TLB. A TLB miss will add 13, 21 or 28 clocks to the instruction depending on whether the Accessed and/or Dirty bit in neither, one or both of the page entries needs to be set in memory. This assumes that neither page entry is in the data cache and a page fault does not occur on the address translation.
11. No exceptions are detected during instruction execution. Refer to Interrupt Clock Counts Table for extra clocks if an interrupt is detected.
12. Instructions that read multiple consecutive data items (i.e. task switch, POPA, etc.) and miss the cache are assumed to start the first access on a 16-byte boundary. If not, an extra cache line fill may be necessary which may add up to (r+3b) clocks to the cache miss penalty.

Table 10.1. Intel486™ Microprocessor Integer Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS				
MOV = Move:				
reg1 to reg2	1000100W 11 reg1 reg2	1		
reg2 to reg1	1000101w 11 reg1 reg2	1		
memory to reg	1000101w mod reg r/m	1	2	
reg to memory	1000100w mod reg r/m	1		
Immediate to reg	1100011w 11000 reg immediate data	1		
or	1011w reg immediate data	1		
Immediate to Memory	1100011w mod 000 r/m displacement immediate	1		
Memory to Accumulator	1010000w full displacement	1	2	
Accumulator to Memory	1010001w full displacement	1		
MOVSX/MOVZX = Move with Sign/Zero Extension				
reg2 to reg1	00001111 1011z11w 11 reg1 reg2	3		
memory to reg	00001111 1011z11w mod reg r/m	3	2	
PUSH = Push				
reg	11111111 11 110 reg	4		
or	01010 reg	1		
memory	11111111 mod 110 r/m	4	1	1
immediate	011010s0 immediate data	1		
PUSHA = Push All	01100000	11		
POP = Pop				
reg	10001111 11 000 reg	4	1	
or	01011 reg	1	2	
memory	10001111 mod 000 r/m	5	2	1
POPA = Pop All	01100001	9	7/15	16/32
XCHG = Exchange				
reg1 with reg2	1000011w 11 reg1 reg2	3		2
Accumulator with reg	10010 reg	3		2
Memory with reg	1000011w mod reg r/m	5		2
NOP = No Operation	10010000	1		
LEA = Load EA to Register	10001101 mod reg r/m			
no index register		1		
with index register		2		

z	Instruction
0	MOVZX
1	MOVSX

2

Table 10.1. Intel486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
Instruction / **TTT**: ADD = Add: 000; ADC = Add with Carry: 010; AND = Logical AND: 100; OR = Logical OR: 001; SUB = Subtract: 101; SBB = Subtract with Borrow: 011; XOR = Logical Exclusive OR: 110				
reg1 to reg2	00TTT00w 11 reg1 reg2	1		
reg2 to reg1	00TTT01w 11 reg1 reg2	1		
memory to register	00TTT01w mod reg r/m	2	2	
register to memory	00TTT00w mod reg r/m	3	6/2	U/L
immediate to register	100000sw 11 TTT reg immediate register	1		
immediate to accumulator	00TTT10w immediate data	1		
immediate to memory	100000sw mod TTT r/m immediate data	3	6/2	U/L
Instruction / **TTT**: INC = Increment: 000; DEC = Decrement: 001				
reg	1111111w 11 TTT reg	1		
or	01TTT reg	1		
memory	1111111w mod TTT r/m	3	6/2	U/L
Instruction / **TTT**: NOT = Logical Complement: 010; NEG = Negate: 011				
reg	1111011w 11 TTT reg	1		
memory	1111011w mod TTT r/m	3	6/2	U/L
CMP = Compare				
reg1 with reg2	0011100w 11 reg1 reg2	1		
reg2 with reg1	0011101w 11 reg1 reg2	1		
memory with register	0011100w mod reg r/m	2	2	
register with memory	0011101w mod reg r/m	2	2	
immediate with register	100000sw 11 111 reg immediate data	1		
immediate with acc.	0011110w immediate data	1		
immediate with memory	100000sw mod 111 r/m immediate data	2	2	
TEST = Logical Compare				
reg1 and reg2	1000010w 11 reg1 reg2	1		
memory and register	1000010w mod reg r/m	2	2	
immediate and register	1111011w 11 000 reg immediate data	1		
immediate and acc.	1010100w immediate data	1		
immediate and memory	1111011w mod 000 r/m immediate data	2	2	

PRELIMINARY

Table 10.1. Intel486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
MUL = Multiply (unsigned)				
acc. with register	1111011w | 11 100 reg			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
acc. with memory	1111011w | mod 100 r/m			
Multiplier-Byte		13/18	1	MN/MX, 3
Word		13/26	1	MN/MX, 3
Dword		13/42	1	MN/MX, 3
IMUL = Integer Multiply (signed)				
acc. with register	1111011w | 11 101 reg			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
acc. with memory	1111011w | mod 101 r/m			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
reg1 with reg2	00001111 | 10101111 | 11 reg1 reg2			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
register with memory	00001111 | 10101111 | mod reg r/m			
Multiplier-Byte		13/18	1	MN/MX, 3
Word		13/26	1	MN/MX, 3
Dword		13/42	1	MN/MX, 3
reg1 with imm. to reg2	011010s1 | 11 reg1 reg2 | immediate data			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
mem. with imm. to reg.	011010s1 | mod reg r/m | immediate data			
Multiplier-Byte		13/18	2	MN/MX, 3
Word		13/26	2	MN/MX, 3
Dword		13/42	2	MN/MX, 3
DIV = Divide (unsigned)				
acc. by register	1111011w | 11 110 reg			
Divisor-Byte		16		
Word		24		
Dword		40		
acc. by memory	1111011w | mod 110 r/m			
Divisor-Byte		16		
Word		24		
Dword		40		
IDIV = Integer Divide (signed)				
acc. by register	1111011w | 11 111 reg			
Divisor-Byte		19		
Word		27		
Dword		43		

2

Table 10.1. Intel486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
acc. by memory	1111011w mod111 r/m			
Divisor-Byte		20		
Word		28		
Dword		44		
CBW/CWD = Convert Byte to Word/ Convert Word to Dword	10011000	3		
CWD/CDQ = Convert Word to Dword/ Convert Dword to Quad Word	10011001	3		

Instruction	TTT
ROL = Rotate Left	000
ROR = Rotate Right	001
RCL = Rotate through Carry Left	010
RCR = Rotate through Carry Right	011
SHL/SAL = Shift Logical/Arithmetic Left	100
SHR = Shift Logical Right	101
SAR = Shift Arithmetic Right	111

Not Through Carry (ROL, ROR, SAL, SAR, SHL, and SHR)				
reg by 1	1101000w 11 TTT reg	3		
memory by 1	1101000w mod TTT r/m	4	6	
reg by CL	1101001w 11 TTT reg	3		
memory by CL	1101001w mod TTT r/m	4	6	
reg by immediate count	1100000w 11 TTT reg immediate 8-bit data	2		
mem by immediate count	1100000w mod TTT r/m immediate 8-bit data	4	6	
Through Carry (RCL and RCR)				
reg by 1	1101000w 11 TTT reg	3		
memory by 1	1101000w mod TTT r/m	4	6	
reg by CL	1101001w 11 TTT reg	8/30		MN/MX, 4
memory by CL	1101001w mod TTT r/m	9/31		MN/MX, 5
reg by immediate count	1100000w 11 TTT reg immediate 8-bit data	8/30		MN/MX, 4
mem by immediate count	1100000w mod TTT r/m immediate 8-bit data	9/31		MN/MX, 5

Instruction	TTT
SHLD = Shift Left Double	100
SHRD = Shift Right Double	101

register with immediate	00001111 10TTT100 11 reg2 reg1 imm 8-bit data	2		
memory by immediate	00001111 10TTT100 mod reg r/m imm 8-bit data	3	6	
register by CL	00001111 10TTT101 11 reg2 reg1	3		
memory by CL	00001111 10TTT101 mod reg r/m	4	5	
BSWAP = Byte Swap	00001111 11001 reg	1		
XADD = Exchange and Add				
reg1, reg2	00001111 1100000w 11 reg2 reg1	3		
memory, reg	00001111 1100000w mod reg r/m	4	6/2	U/L
CMPXCHG = Compare and Exchange				
reg1, reg2	00001111 1011000w 11 reg2 reg1	6		
memory, reg	00001111 1011000w mod reg r/m	7/10	2	6

Table 10.1. Intel486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
CONTROL TRANSFER (within segment)				
NOTE: Times are jump taken/not taken				
Jccc = Jump on ccc				
8-bit displacement	0111tttn | 8-bit disp.	3/1		T/NT, 23
full displacement	00001111 | 1000tttn | full displacement	3/1		T/NT, 23
NOTE: Times are jump taken/not taken				
SETcccc = Set Byte on cccc (Times are cccc true/false)				
reg	00001111 | 1001tttn | 11 000 reg	4/3		
memory	00001111 | 1001tttn | mod 000 r/m	3/4		

Mnemonic cccc	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less Than or Equal/Greater Than	1111

LOOP = LOOP CX Times	11100010 | 8-bit disp.	7/6		L/NL, 23
LOOPZ/LOOPE = Loop with Zero/Equal	11100001 | 8-bit disp.	9/6		L/NL, 23
LOOPNZ/LOOPNE = Loop while Not Zero	11100000 | 8-bit disp.	9/6		L/NL, 23
JCXZ = Jump on CX Zero	11100011 | 8-bit disp.	8/5		T/NT, 23
JECXZ = Jump on ECX Zero	11100011 | 8-bit disp.	8/5		T/NT, 23
(Address Size Prefix Differentiates JCXZ for JECXZ)				
JMP = Unconditional Jump (within segment)				
Short	11101011 | 8-bit disp.	3		7, 23
Direct	11101001 | full displacement	3		7, 23
Register Indirect	11111111 | 11 100 reg	5		7, 23
Memory Indirect	11111111 | mod 100 r/m	5	5	7
CALL = Call (within segment)				
Direct	11101000 | full displacement	3		7, 23
Register Indirect	11111111 | 11 010 reg	5		7, 23
Memory Indirect	11111111 | mod 010 r/m	5	5	7
RET = Return from CALL (within segment)				
	11000011	5	5	
Adding Immediate to SP	11000010 | 16-bit disp.	5	5	

Table 10.1. Intel486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
CONTROL TRANSFER (within segment) (Continued)				
ENTER = Enter Procedure	11001000 | 16-bit disp., 8-bit level			
Level = 0		14		
Level = 1		17		
Level (L) > 1		17+3L		8
LEAVE = Leave Procedure	11001001	5	1	
MULTIPLE-SEGMENT INSTRUCTIONS				
MOV = Move				
reg. to segment reg.	10001110 | 11 sreg3 reg	3/9	0/3	RV/P, 9
memory to segment reg.	10001110 | mod sreg3 r/m	3/9	2/5	RV/P, 9
segment reg. to reg.	10001100 | 11 sreg3 reg	3		
segment reg. to memory	10001100 | mod sreg3 r/m	3		
PUSH = Push				
segment reg. (ES, CS, SS, or DS)	000sreg2110	3		
segment reg. (FS or GS)	00001111 | 10 sreg3000	3		
POP = Pop				
segment reg. (ES, SS, or DS)	000sreg2111	3/9	2/5	RV/P, 9
segment reg. (FS or GS)	00001111 | 10 sreg3001	3/9	2/5	RV/P, 9
LDS = Load Pointer to DS	11000101 | mod reg r/m	6/12	7/10	RV/P, 9
LES = Load Pointer to ES	11000100 | mod reg r/m	6/12	7/10	RV/P, 9
LFS = Load Pointer to FS	00001111 | 10110100 | mod reg r/m	6/12	7/10	RV/P, 9
LGS = Load Pointer to GS	00001111 | 10110101 | mod reg r/m	6/12	7/10	RV/P, 9
LSS = Load Pointer to SS	00001111 | 10110010 | mod reg r/m	6/12	7/10	RV/P, 9
CALL = Call				
Direct intersegment	10011010 | unsigned full offset, selector	18	2	R, 7, 22
to same level		20	3	P, 9
thru Gate to same level		35	6	P, 9
to inner level, no parameters		69	17	P, 9
to inner level, x parameter (d) words		77+4X	17+n	P, 11, 9
to TSS		37+TS	3	P, 10, 9
thru Task Gate		38+TS	3	P, 10, 9
Indirect intersegment	11111111 | mod 011 r/m	17	8	R, 7
to same level		20	10	P, 9
thru Gate to same level		35	13	P, 9
to inner level, no parameters		69	24	P, 9
to inner level, x parameter (d) words		77+4X	24+n	P, 11, 9
to TSS		37+TS	10	P, 10, 9
thru Task Gate		38+TS	10	P, 10, 9
RET = Return from CALL				
intersegment	11001011	13	8	R, 7
to same level		17	9	P, 9
to outer level		35	12	P, 9
intersegment adding imm. to SP	11001010 | 16-bit disp.	14	8	R, 7
to same level		18	9	P, 9
to outer level		36	12	P, 9

PRELIMINARY

Table 10.1. Intel486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
MULTIPLE-SEGMENT INSTRUCTIONS (Continued)				
JMP = Unconditional Jump				
Direct intersegment	11101010 | unsigned full offset, selector	17	2	R, 7, 22
to same level		19	3	P, 9
thru Call Gate to same level		32	6	P, 9
thru TSS		42+TS	3	P, 10, 9
thru Task Gate		43+TS	3	P, 10, 9
Indirect intersegment	11111111 | mod 101 r/m	13	9	R, 7, 9
to same level		18	10	P, 9
thru Call Gate to same level		31	13	P, 9
thru TSS		41+TS	10	P, 10, 9
thru Task Gate		42+TS	10	P, 10, 9
BIT MANIPULATION				
BT = Test bit				
register, immediate	00001111 | 10111010 | 11 100 reg | imm. 8-bit data	3		
memory, immediate	00001111 | 10111010 | mod 100 r/m | imm. 8-bit data	3	1	
reg1, reg2	00001111 | 10100011 | 11 reg2 reg1	3		
memory, reg	00001111 | 10100011 | mod reg r/m	8	2	
Instruction / **TTT**				
BTS = Test Bit and Set	101			
BTR = Test Bit and Reset	110			
BTC = Test Bit and Compliment	111			
register, immediate	00001111 | 10111010 | 11 TTT reg | imm. 8-bit data	6		
memory, immediate	00001111 | 10111010 | mod TTT r/m | imm. 8-bit data	8	2/0	U/L
reg1, reg2	00001111 | 10TTT011 | 11 reg2 reg1	6		
memory, reg	00001111 | 10TTT011 | mod reg r/m	13	3/1	U/L
BSF = Scan Bit Forward				
reg1, reg2	00001111 | 10111100 | 11 reg2 reg1	6/42		MN/MX, 12
memory, reg	00001111 | 10111100 | mod reg r/m	7/43	2	MN/MX, 13
BSR = Scan Bit Reverse				
reg1, reg2	00001111 | 10111101 | 11 reg2 reg1	6/103		MN/MX, 14
memory, reg	00001111 | 10111101 | mod reg r/m	7/104	1	MN/MX, 15
STRING INSTRUCTIONS				
CMPS = Compare Byte Word	1010011w	8	6	16
LODS = Load Byte/Word to AL/AX/EAX	1010110w	5	2	
MOVS = Move Byte/Word	1010010w	7	2	16
SCAS = Scan Byte/Word	1010111w	6	2	
STOS = Store Byte/Word from AL/AX/EX	1010101w	5		
XLAT = Translate String	11010111	4	2	

2

Table 10.1. Intel486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
REPEATED STRING INSTRUCTIONS				
Repeated by Count in CX or ECX (C = Count in CX or ECX)				
REPE CMPS = Compare String (Find Non-Match)	11110011 1010011w			
C = 0		5		
C > 0		7+7c		16, 17
REPNE CMPS = Compare String (Find Match)	11110010 1010011w			
C = 0		5		
C > 0		7+7c		16, 17
REP LODS = Load String	11110011 1010110w			
C = 0		5		
C > 0		7+4c		16, 18
REP MOVS = Move String	11110011 1010010w			
C = 0		5		
C = 1		13	1	16
C > 1		12+3c		16, 19
REPE SCAS = Scan String (Find Non-AL/AX/EAX)	11110011 1010111w			
C = 0		5		
C > 0		7+5c		20
REPNE SCAS = Scan String (Find AL/AX/EAX)	11110010 1010111w			
C = 0		5		
C > 0		7+5c		20
REP STOS = Store String	11110011 1010101w			
C = 0		5		
C > 0		7+4c		
FLAG CONTROL				
CLC = Clear Carry Flag	11111000	2		
STC = Set Carry Flag	11111001	2		
CMC = Complement Carry Flag	11110101	2		
CLD = Clear Direction Flag	11111100	2		
STD = Set Direction Flag	11111101	2		
CLI = Clear Interrupt Enable Flag	11111010	5		
STI = Set Interrupt Enable Flag	11111011	5		
LAHF = Load AH into Flag	10011111	3		
SAHF = Store AH into Flags	10011110	2		
PUSHF = Push Flags	10011100	4/3		RV/P
POPF = Pop Flags	10011101	9/6		RV/P
DECIMAL ARITHMETIC				
AAA = ASCII Adjust for Add	00110111	3		
AAS = ASCII Adjust for Subtract	00111111	3		
AAM = ASCII Adjust for Multiply	11010100 00001010	15		

Table 10.1. Intel486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty If Cache Miss	Notes
DECIMAL ARITHMETIC (Continued)				
AAD = ASCII Adjust for Divide	11010101 00001010	14		
DAA = Decimal Adjust for Add	00100111	2		
DAS = Decimal Adjust for Subtract	00101111	2		
PROCESSOR CONTROL INSTRUCTIONS				
HLT = Halt	11110100	4		
MOV = Move To and From Control/Debug/Test Registers				
CR0 from register	00001111 00100010 11 000 reg	17	2	
CR2/CR3 from register	00001111 00100010 11 eee reg	4		
Reg from CR0–3	00001111 00100000 11 eee reg	4		
DR0–3 from register	00001111 00100011 11 eee reg	10		
DR6–7 from register	00001111 00100011 11 eee reg	10		
Register from DR6–7	00001111 00100001 11 eee reg	9		
Register from DR0–3	00001111 00100001 11 eee reg	9		
TR3 from register	00001111 00100110 11 011 reg	4		
TR4–7 from register	00001111 00100110 11 eee reg	4		
Register from TR3	00001111 00100100 11 011 reg	3		
Register from TR4–7	00001111 00100100 11 eee reg	4		
CLTS = Clear Task Switched Flag	00001111 00000110	7	2	
INVD = Invalidate Data Cache	00001111 00001000	4		
WBINVD = Write-Back and Invalidate Data Cache	00001111 00001001	5		
INVLPG = Invalidate TLB Entry				
INVLPG memory	00001111 00000001 mod 111 r/m	12/11		H/NH
PREFIX BYTES				
Address Size Prefix	01100111	1		
LOCK = Bus Lock Prefix	11110000	1		
Operand Size Prefix	01100110	1		
Segment Override Prefix				
CS:	00101110	1		
DS:	00111110	1		
ES:	00100110	1		
FS:	01100100	1		
GS:	01100101	1		
SS:	00110110	1		

Table 10.1. Intel486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
PROTECTION CONTROL				
ARPL = Adjust Requested Privilege Level				
From register	01100011 | 11 reg1 reg2	9		
From memory	01100011 | mod reg r/m	9		
LAR = Load Access Rights				
From register	00001111 | 00000010 | 11 reg1 reg2	11	3	
From memory	00001111 | 00000010 | mod reg r/m	11	5	
LGDT = Load Global Descriptor				
Table register	00001111 | 00000001 | mod 010 r/m	12	5	
LIDT = Load Interrupt Descriptor				
Table register	00001111 | 00000001 | mod 011 r/m	12	5	
LLDT = Load Local Descriptor				
Table register from reg.	00001111 | 00000000 | 11 010 reg	11	3	
Table register from mem.	00001111 | 00000000 | mod 010 r/m	11	6	
LMSW = Load Machine Status Word				
From register	00001111 | 00000001 | 11 110 reg	13		
From memory	00001111 | 00000001 | mod 110 r/m	13	1	
LSL = Load Segment Limit				
From register	00001111 | 00000011 | 11 reg1 reg2	10	3	
From memory	00001111 | 00000011 | mod reg r/m	10	6	
LTR = Load Task Register				
From Register	00001111 | 00000000 | 11 011 reg	20		
From Memory	00001111 | 00000000 | mod 011 r/m	20		
SGDT = Store Global Descriptor Table				
	00001111 | 00000001 | mod 000 r/m	10		
SIDT = Store Interrupt Descriptor Table				
	00001111 | 00000001 | mod 001 r/m	10		
SLDT = Store Local Descriptor Table				
To register	00001111 | 00000000 | 11 000 reg	2		
To memory	00001111 | 00000000 | mod 000 r/m	3		
SMSW = Store Machine Status Word				
To register	00001111 | 00000001 | 11 100 reg	2		
To memory	00001111 | 00000001 | mod 100 r/m	3		
STR = Store Task Register				
To register	00001111 | 00000000 | 11 001 reg	2		
To memory	00001111 | 00000000 | mod 001 r/m	3		
VERR = Verify Read Access				
Register	00001111 | 00000000 | 11 100 r/m	11	3	
Memory	00001111 | 00000000 | mod 100 r/m	11	7	
VERW = Verify Write Access				
To register	00001111 | 00000000 | 11 101 reg	11	3	
To memory	00001111 | 00000000 | mod 101 r/m	11	7	

Table 10.1. Intel486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTERRUPT INSTRUCTIONS				
INT n = Interrupt Type n	11001101 | type	INT+4/0		RV/P, 21
INT 3 = Interrupt Type 3	11001100	INT+0		21
INTO = Interrupt 4 if Overflow Flag Set	11001110			
Taken		INT+2		21
Not Taken		3		21
BOUND = Interrupt 5 if Detect Value Out Range	01100010 | mod reg r/m			
If in range		7	7	21
If out of range		INT+24	7	21
IRET = Interrupt Return	11001111			
Real Mode/Virtual Mode		15	8	
Protected Mode				
To same level		20	11	9
To outer level		36	19	9
To nested task (EFLAGS.NT = 1)		TS+32	4	9, 10
External Interrupt		INT+11		21
NMI = Non-Maskable Interrupt		INT+3		21
Page Fault		INT+24		21
VM86 Exceptions				
CLI		INT+8		21
STI		INT+8		21
INT n		INT+9		
PUSHF		INT+9		21
POPF		INT+8		21
IRET		INT+9		
IN				
Fixed Port		INT+50		21
Variable Port		INT+51		21
OUT				
Fixed Port		INT+50		21
Variable Port		INT+51		21
INS		INT+50		21
OUTS		INT+50		21
REP INS		INT+51		21
REP OUTS		INT+51		21

Task Switch Clock Counts Table		
Method	**Value for TS**	
	Cache Hit	**Miss Penalty**
VM/Intel486 CPU/286 TSS To Intel486 CPU TSS	162	55
VM/Intel486 CPU/286 TSS To 286 TSS	143	31
VM/Intel486 CPU/286 TSS To VM TSS	140	37

Interrupt Clock Counts Table			
Method	Value for INT		
	Cache Hit	Miss Penalty	Notes
Real Mode	26	2	
Protected Mode			
Interrupt/Trap gate, same level	44	6	9
Interrupt/Trap gate, different level	71	17	9
Task Gate	37 + TS	3	9, 10
Virtual Mode			
Interrupt/Trap gate, different level	82	17	
Task gate	37 + TS	3	10

Abbreviations	Definition
16/32	16/32 bit modes
U/L	unlocked/locked
MN/MX	minimum/maximum
L/NL	loop/no loop
RV/P	real and virtual mode/protected mode
R	real mode
P	protected mode
T/NT	taken/not taken
H/NH	hit/no hit

NOTES:

1. Assuming that the operand address and stack address fall in different cache sets.
2. Always locked, no cache hit case.
3. Clocks = 10 + max($\log_2$(|m|),n)
   m = multiplier value (min clocks for m=0)
   n = 3/5 for ±m
4. Clocks = {quotient(count/operand length)}*7+9
   = 8 if count ≤ operand length (8/16/32)
5. Clocks = {quotient(count/operand length)}*7+9
   = 9 if count ≤ operand length (8/16/32)
6. Equal/not equal cases (penalty is the same regardless of lock).
7. Assuming that addresses for memory read (for indirection), stack push/pop, and branch fall in different cache sets.
8. Penalty for cache miss: add 6 clocks for every 16 bytes copied to new stack frame.
9. Add 11 clocks for each unaccessed descriptor load.
10. Refer to task switch clock counts table for value of TS.
11. Add 4 extra clocks to the cache miss penalty for each 16 bytes.

For notes 12–13: (b = 0–3, non-zero byte number);
(i = 0–1, non-zero nibble number);
(n = 0–3, non bit number in nibble);

12. Clocks = 8+4 (b+1) + 3(i+1) + 3(n+1)
    = 6 if second operand = 0
13. Clocks = 9+4(b+1) + 3(i+1) + 3(n+1)
    = 7 if second operand = 0 .

For notes 14–15: (n = bit position 0–31)

14. Clocks = 7 + 3(32−n)
    6 if second operand = 0
15. Clocks = 8 + 3(32−n)
    7 if second operand = 0
16. Assuming that the two string addresses fall in different cache sets.
17. Cache miss penalty: add 6 clocks for every 16 bytes compared. Entire penalty on first compare.
18. Cache miss penalty: add 2 clocks for every 16 bytes of data. Entire penalty on first load.
19. Cache miss penalty: add 4 clocks for every 16 bytes moved.
    (1 clock for the first operation and 3 for the second)
20. Cache miss penalty: add 4 clocks for every 16 bytes scanned.
    (2 clocks each for first and second operations)
21. Refer to interrupt clock counts table for value of INT
22. Clock count includes one clock for using both displacement and immediate.
23. Refer to assumption 6 in the case of a cache miss.

PRELIMINARY

Table 10.2. Intel486™ Microprocessor I/O Instructions Clock Count Summary

INSTRUCTION	FORMAT	Real Mode	Protected Mode (CPL≤IOPL)	Protected Mode (CPL>IOPL)	Virtual 86 Mode	Notes
I/O INSTRUCTIONS						
IN = Input from:						
Fixed Port	1110010w | port number	14	9	29	27	
Variable Port	1110110w	14	8	28	27	
OUT = Output to:						
Fixed Port	1110011w | port number	16	11	31	29	
Variable Port	1110111w	16	10	30	29	
INS = Input Byte/Word from DX Port	0110110w	17	10	32	30	
OUTS = Output Byte/Word to DX Port	0110111w	17	10	32	30	1
REP INS = Input String	11110011 | 0110110w	16+8c	10+8c	30+8c	29+8c	2
REP OUTS = Output String	11110011 | 0110111w	17+5c	11+5c	31+5c	30+5c	3

NOTES:
1. Two clock cache miss penalty in all cases.
2. c = count in CX or ECX.
3. Cache miss penalty in all modes: Add 2 clocks for every 16 bytes. Entire penalty on second operation.

2

Table 10.3. Intel486™ Microprocessor Floating Point Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range . . . Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range . . . Upper Range)	Notes
DATA TRANSFER					
FLD = Real Load to ST(0)					
32-bit memory	11011 001 | mod 000 r/m | s-i-b/disp.	3	2		
64-bit memory	11011 101 | mod 000 r/m | s-i-b/disp.	3	3		
80-bit memory	11011 011 | mod 101 r/m | s-i-b/disp.	6	4		
ST(i)	11011 001 | 11000 ST(i)	4			
FILD = Integer Load to ST(0)					
16-bit memory	11011 111 | mod 000 r/m | s-i-b/disp.	14.5(13–16)	2	4	
32-bit memory	11011 011 | mod 000 r/m | s-i-b/disp.	11.5(9–12)	2	4(2–4)	
64-bit memory	11011 111 | mod 101 r/m | s-i-b/disp.	16.8(10–18)	3	7.8(2–8)	
FBLD = BCD Load to ST(0)	11011 111 | mod 100 r/m | s-i-b/disp.	75(70–103)	4	7.7(2–8)	
FST = Store Real from ST(0)					
32-bit memory	11011 001 | mod 010 r/m | s-i-b/disp.	7			1
64-bit memory	11011 101 | mod 010 r/m | s-i-b/disp.	8			2
ST(i)	11011 101 | 11010 ST(i)	3			
FSTP = Store Real from ST(0) and Pop					
32-bit memory	11011 001 | mod 011 r/m | s-i-b/disp.	7			1
64-bit memory	11011 101 | mod 011 r/m | s-i-b/disp.	8			2
80-bit memory	11011 011 | mod 111 r/m | s-i-b/disp.	6			
ST(i)	11011 101 | 11001 ST(i)	3			
FIST = Store Integer from ST(0)					
16-bit memory	11011 111 | mod 010 r/m | s-i-b/disp.	33.4(29–34)			
32-bit memory	11011 011 | mod 010 r/m | s-i-b/disp.	32.4(28–34)			
FISTP = Store Integer from ST(0) and Pop					
16-bit memory	11011 111 | mod 011 r/m | s-i-b/disp.	33.4(29–34)			
32-bit memory	11011 011 | mod 011 r/m | s-i-b/disp.	33.4(29–34)			
64-bit memory	11011 111 | mod 111 r/m | s-i-b/disp.	33.4(29–34)			
FBSTP = Store BCD from ST(0) and Pop	11011 111 | mod 110 r/m | s-i-b/disp.	175(172–176)			
FXCH = Exchange ST(0) and ST(i)	11011 001 | 11001 ST(i)	4			
COMPARISON INSTRUCTIONS					
FCOM = Compare ST(0) with Real					
32-bit memory	11011 000 | mod 010 r/m | s-i-b/disp.	4	2	1	
64-bit memory	11011 100 | mod 010 r/m | s-i-b/disp.	4	3	1	
ST(i)	11011 000 | 11010 ST(i)	4		1	
FCOMP = Compare ST(0) with Real and Pop					
32-bit memory	11011 000 | mod 011 r/m | s-i-b/disp.	4	2	1	
64-bit memory	11011 100 | mod 011 r/m | s-i-b/disp.	4	3	1	
ST(i)	11011 000 | 11011 ST(i)	4		1	

Table 10.3. Intel486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit: Avg (Lower Range ... Upper Range)	Penalty if Cache Miss	Concurrent Execution: Avg (Lower Range ... Upper Range)	Notes
COMPARISON INSTRUCTIONS (Continued)					
FCOMPP = Compare ST(0) with ST(1) and Pop Twice	11011 110 | 1101 1001	5		1	
FICOM = Compare ST(0) with Integer					
16-bit memory	11011 110 | mod 010 r/m | s-i-b/disp.	18(16–20)	2	1	
32-bit memory	11011 010 | mod 010 r/m | s-i-b/disp.	16.5(15–17)	2	1	
FICOMP = Compare ST(0) with Integer					
16-bit memory	11011 110 | mod 011 r/m | s-i-b/disp.	18(16–20)	2	1	
32-bit memory	11011 010 | mod 011 r/m | s-i-b/disp.	16.5(15–17)	2	1	
FTST = Compare ST(0) with 0.0	11011 001 | 1110 0100	4		1	
FUCOM = Unordered compare ST(0) with ST(i)	11011 101 | 11100 ST(i)	4		1	
FUCOMP = Unordered compare ST(0) with ST(i) and Pop	11011 101 | 11101 ST(i)	4		1	
FUCOMPP = Unordered compare ST(0) with ST(i) and Pop Twice	11011 010 | 1110 1001	5		1	
FXAM = Examine ST(0)	11011 001 | 1110 0101	8			
CONSTANTS					
FLDZ = Load +0.0 into ST(0)	11011 001 | 1110 1110	4			
FLD1 = Load +1.0 into ST(0)	11011 001 | 1110 1000	4			
FLDPI = Load π into ST(0)	11011 001 | 1110 1011	8		2	
FLDL2T = Load $\log_2(10)$ into ST(0)	11011 001 | 1110 1001	8		2	
FLDL2E = Load $\log_2(e)$ into ST(0)	11011 001 | 1110 1010	8		2	
FLDLG2 = Load $\log_{10}(2)$ into ST(0)	11011 001 | 1110 1100	8		2	
FLDLN2 = Load $\log_e(2)$ into ST(0)	11011 001 | 1110 1101	8		2	
ARITHMETIC					
FADD = Add Real with ST(0)					
ST(0) ← ST(0) + 32-bit memory	11011 000 | mod 000 r/m | s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0) ← ST(0) + 64-bit memory	11011 100 | mod 000 r/m | s-i-b/disp.	10(8–20)	3	7(5–17)	
ST(d) ← ST(0) + ST(i)	11011 d00 | 11000 ST(i)	10(8–20)		7(5–17)	
FADDP = Add real with ST(0) and Pop (ST(i) ← ST(0) + ST(i))	11011 110 | 11000 ST(i)	10(8–20)		7(5–17)	
FSUB = Subtract real from ST(0)					
ST(0) ← ST(0) − 32-bit memory	11011 000 | mod 100 r/m | s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0) ← ST(0) − 64-bit memory	11011 100 | mod 100 r/m | s-i-b/disp.	10(8-20)	3	7(5–17)	
ST(d) ← ST(0) − ST(i)	11011 d00 | 1110d ST(i)	10(8–20)		7(5–17)	
FSUBP = Subtract real from ST(0) and Pop (ST(i) ← ST(0) − ST(i))	11011 110 | 11101 ST(i)	10(8–20)		7(5–17)	

Table 10.3. Intel486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit: Avg (Lower Range ... Upper Range)	Penalty if Cache Miss	Concurrent Execution: Avg (Lower Range ... Upper Range)	Notes
ARITHMETIC (Continued)					
FSUBR = Subtract real reversed (Subtract ST(0) from real)					
ST(0) ← 32-bit memory − ST(0)	11011 000 | mod 101 r/m | s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0) ← 64-bit memory − ST(0)	11011 100 | mod 101 r/m | s-i-b/disp.	10(8–20)	3	7(5–17)	
ST(d) ← ST(i) − ST(0)	11011 d00 | 1110d ST(i)	10(8–20)		7(5–17)	
FSUBRP = Subtract real reversed and Pop (ST(i) ← ST(i) − ST(0))	11011 110 | 11100 ST(i)	10(8–20)		7(5–17)	
FMUL = Multiply real with ST(0)					
ST(0) ← ST(0) × 32-bit memory	11011 000 | mod 001 r/m | s-i-b/disp.	11	2	8	
ST(0) ← ST(0) × 64-bit memory	11011 100 | mod 001 r/m | s-i-b/disp.	14	3	11	
ST(d) ← ST(0) × ST(i)	11011 d00 | 11001 ST(i)	16		13	
FMULP = Multiply ST(0) with ST(i) and Pop (ST(i) ← ST(0) × ST(i))	11011 110 | 11001 ST(i)	16		13	
FDIV = Divide ST(0) by Real					
ST(0) ← ST(0)/32-bit memory	11011 000 | mod 110 r/m | s-i-b/disp.	73	2	70	3
ST(0) ← ST(0)/64-bit memory	11011 100 | mod 110 r/m | s-i-b/disp.	73	3	70	3
ST(d) ← ST(0)/ST(i)	11011 d00 | 1111d ST(i)	73		70	3
FDIVP = Divide ST(0) by ST(i) and Pop (ST(i) ← ST(0)/ST(i))	11011 110 | 11111 ST(i)	73		70	3
FDIVR = Divide real reversed (Real/ST(0))					
ST(0) ← 32-bit memory/ST(0)	11011 000 | mod 111 r/m | s-i-b/disp.	73	2	70	3
ST(0) ← 64-bit memory/ST(0)	11011 100 | mod 111 r/m | s-i-b/disp.	73	3	70	3
ST(d) ← ST(i)/ST(0)	11011 d00 | 1111d ST(i)	73		70	3
FDIVRP = Divide real reversed and Pop (ST(i) ← ST(i)/ST(0))	11011 110 | 11110 ST(i)	73		70	3
FIADD = Add Integer to ST(0)					
ST(0) ← ST(0) + 16-bit memory	11011 110 | mod 000 r/m | s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0) ← ST(0) + 32-bit memory	11011 010 | mod 000 r/m | s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FISUB = Subtract Integer from ST(0)					
ST(0) ← ST(0) − 16-bit memory	11011 110 | mod 100 r/m | s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0) ← ST(0) − 32-bit memory	11011 010 | mod 100 r/m | s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FISUBR = Integer Subtract Reversed					
ST(0) ← 16-bit memory − ST(0)	11011 110 | mod 101 r/m | s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0) ← 32-bit memory − ST(0)	11011 010 | mod 101 r/m | s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FIMUL = Multiply Integer with ST(0)					
ST(0) ← ST(0) × 16-bit memory	11011 110 | mod 001 r/m | s-i-b/disp.	25(23–27)	2	8	
ST(0) ← ST(0) × 32-bit memory	11011 010 | mod 001 r/m | s-i-b/disp.	23.5(22–24)	2	8	
FIDIV = Integer Divide					
ST(0) ← ST(0)/16-bit memory	11011 110 | mod 110 r/m | s-i-b/disp.	87(85–89)	2	70	3
ST(0) ← ST(0)/32-bit memory	11011 010 | mod 110 r/m | s-i-b/disp.	85.5(84–86)	2	70	3

Table 10.3. Intel486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range . . . Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range . . . Upper Range)	Notes
ARITHMETIC (Continued)					
FIDIVR = Integer Divide Reversed					
ST(0) ← 16-bit memory/ST(0)	11011 110 | mod 111 r/m | s-i-b/disp.	87(85–89)	2	70	3
ST(0) ← 32-bit memory/ST(0)	11011 010 | mod 111 r/m | s-i-b/disp.	85.5(84–86)	2	70	3
FSQRT = Square Root	11011 001 | 1111 1010	85.5(83–87)		70	
FSCALE = Scale ST(0) by ST(1)	11011 001 | 1111 1101	31(30–32)		2	
FXTRACT = Extract components of ST(0)	11011 001 | 1111 0100	19(16–20)		4(2–4)	
FPREM = Partial Reminder	11011 001 | 1111 1000	84(70–138)		2(2–8)	
FPREM1 = Partial Reminder (IEEE)	11011 001 | 1111 0101	94.5(72–167)		5.5(2–18)	
FRNDINT = Round ST(0) to integer	11011 001 | 1111 1100	29.1(21–30)		7.4(2–8)	
FABS = Absolute value of ST(0)	11011 001 | 1110 0001	3			
FCHS = Change sign of ST(0)	11011 001 | 1110 0000	6			
TRANSCENDENTAL					
FCOS = Cosine of ST(0)	11011 001 | 1111 1111	241(193–279)		2	6, 7
FPTAN = Partial tangent of ST(0)	11011 001 | 1111 0010	244(200–273)		70	6, 7
FPATAN = Partial arctangent	11011 001 | 1111 0011	289(218–303)		5(2–17)	6
FSIN = Sine of ST(0)	11011 001 | 1111 1110	241(193–279)		2	6, 7
FSINCOS = Sine and cosine of ST(0)	11011 001 | 1111 1011	291(243–329)		2	6, 7
F2XM1 = $2^{ST(0)} - 1$	11011 001 | 1111 0000	242(140–279)		2	6
FYL2X = ST(1) × $\log_2$(ST(0))	11011 001 | 1111 0001	311(196–329)		13	6
FYL2XP1 = ST(1) × $\log_2$(ST(0) + 1.0)	11011 001 | 1111 1001	313(171–326)		13	6
PROCESSOR CONTROL					
FINIT = Initialize FPU	11011 011 | 1110 0011	17			4
FSTSW AX = Store status word into AX	11011 111 | 1110 0000	3			5
FSTSW = Store status word into memory	11011 101 | mod 111 r/m | s-i-b/disp.	3			5
FLDCW = Load control word	11011 001 | mod 101 r/m | s-i-b/disp.	4	2		
FSTCW = Store control word	11011 001 | mod 111 r/m | s-i-b/disp.	3			5
FCLEX = Clear exceptions	11011 011 | 1110 0010	7			4
FSTENV = Store environment	11011 001 | mod 110 r/m | s-i-b/disp.				
Real and Virtual modes 16-bit Address		67			4
Real and Virtual modes 32-bit Address		67			4
Protected mode 16-bit Address		56			4
Protected mode 32-bit Address		56			4
FLDENV = Load environment	11011 001 | mod 100 r/m | s-i-b/disp.				
Real and Virtual modes 16-bit Address		44	2		
Real and Virtual modes 32-bit Address		44	2		
Protected mode 16-bit Address		34	2		
Protected mode 32-bit Address		34	2		

Table 10.3. Intel486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit: Avg (Lower Range ... Upper Range)	Penalty If Cache Miss	Concurrent Execution: Avg (Lower Range ... Upper Range)	Notes
PROCESSOR CONTROL (Continued)					
FSAVE = Save state	11011 101 | mod 110 r/m | s-i-b/disp.				
Real and Virtual modes 16-bit Address		154			4
Real and Virtual modes 32-bit Address		154			4
Protected mode 16-bit Address		143			4
Protected mode 32-bit Address		143			4
FRSTOR = Restore state	11011 101 | mod 100 r/m | s-i-b/				
Real and Virtual modes 16-bit Address		131	23		
Real and Virtual modes 32-bit Address		131	27		
Protected mode 16-bit Address		120	23		
Protected mode 32-bit Address		120	27		
FINCSTP = Increment Stack Pointer	11011 001 | 1111 0111	3			
FDECSTP = Decrement Stack Pointer	11011 001 | 1111 0110	3			
FFREE = Free ST(i)	11011 101 | 11000 ST(i)	3			
FNOP = No operations	11011 001 | 1101 0000	3			
WAIT = Wait until FPU ready	10011011				
(Minimum/Maximum)		1/3			

NOTES:
1. If operand is 0 clock counts = 27.
2. If operand is 0 clock counts = 28.
3. If CW.PC indicates 24 bit precision then subtract 38 clocks.
   If CW.PC indicates 53 bit precision then subtract 11 clocks.
4. If there is a numeric error pending from a previous instruction add 17 clocks.
5. If there is a numeric error pending from a previous instruction add 18 clocks.
6. The INT pin is polled several times while this instruction is executing to assure short interrupt latency.
7. If ABS(operand) is greater than $\pi/4$ then add n clocks. Where n = (operand/($\pi/4$)).

10.2 Instruction Encoding

10.2.1 OVERVIEW

All instruction encodings are subsets of the general instruction format shown in Figure 10.1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the "mod r/m" byte and "scaled index" byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16 or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 10.1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the Figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 10.4 is a complete list of all fields appearing in the Intel486 Microprocessor instruction set. Further ahead, following Table 10.4, are detailed tables for each field.

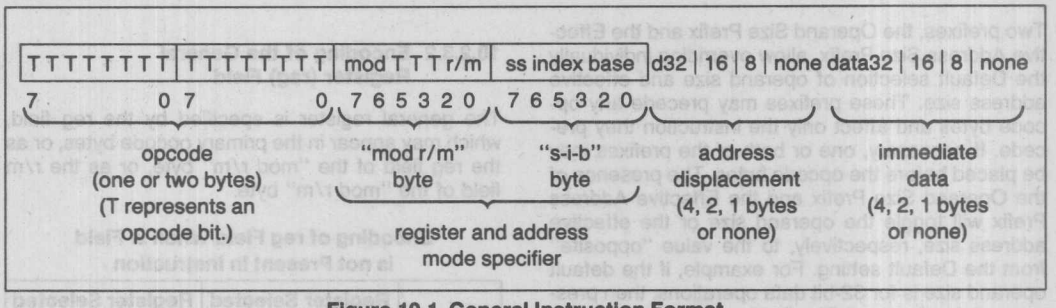


Figure 10.1. General Instruction Format

Table 10.4. Fields within Intel486™ Microprocessor Instructions

Field Name	Description	Number of Bits
w	Specifies if Data is Byte or Full Size (Full Size is either 16 or 32 Bits	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field Must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
tttn	For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated	4

NOTE:
Tables 10.1–10.3 show encoding of individual instructions.

10.2.2 32-BIT EXTENSIONS OF THE INSTRUCTION SET

With the Intel486 Microprocessor, the 8086/80186/80286 instruction set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

Whether the instruction defaults to operations of 16 bits or 32 bits depends on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 bits or 16 bits) for both operands and effective addresses when executing that code segment. In the Real Address Mode or Virtual 8086 Mode, no code segment descriptors are used, but a D value of 0 is assumed internally by the Intel486 Microprocessor when operating in those modes (for 16-bit default sizes compatible with the 8086/80186/80286).

Two prefixes, the Operand Size Prefix and the Effective Address Size Prefix, allow overriding individually the Default selection of operand size and effective address size. These prefixes may precede any opcode bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed before the opcode bytes. The presence of the Operand Size Prefix and the Effective Address Prefix will toggle the operand size or the effective address size, respectively, to the value "opposite" from the Default setting. For example, if the default operand size is for 32-bit data operations, then presence of the Operand Size Prefix toggles the instruction to 16-bit data operation. As another example, if the default effective address size is 16 bits, presence of the Effective Address Size prefix toggles the instruction to use 32-bit effective address computations.

These 32-bit extensions are available in all Intel486 Microprocessor modes, including the Real Address Mode or the Virtual 8086 Mode. In these modes the default is always 16 bits, so prefixes are needed to specify 32-bit operands or addresses. For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

10.2.3 ENCODING OF INTEGER INSTRUCTION FIELDS

Within the instruction are several fields indicating register selection, addressing mode and so on. The exact encodings of these fields are defined immediately ahead.

10.2.3.1 Encoding of Operand Length (w) Field

For any given instruction performing a data operation, the instruction is executing as a 32-bit operation or a 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size During 16-Bit Data Operations	Operand Size During 32-Bit Data Operations
0	8 Bits	8 Bits
1	16 Bits	32 Bits

10.2.3.2 Encoding of the General Register (reg) Field

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the "mod r/m" byte, or as the r/m field of the "mod r/m" byte.

Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected During 16-Bit Data Operations	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations:		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field During 32-Bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

10.2.3.3 Encoding of the Segment Register (sreg) Field

The sreg field in certain instructions is a 2-bit field allowing one of the four 80286 segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the Intel486 Microprocessor FS and GS segment registers to be specified.

2-Bit sreg2 Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

10.2.3.4 Encoding of Address Mode

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the "mod r/m" byte, and a second byte of addressing information, the "s-i-b" (scale-index-base) byte, can be specified.

The s-i-b byte (scale-index-base byte) is specified when using 32-bit addressing mode and the "mod r/m" byte has r/m = 100 and mod = 00, 01 or 10. When the sib byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the "mod r/m" byte, also contains three bits (shown as TTT in Figure 10.1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the "mod r/m" byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the "mod r/m" byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following three pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Encoding of 16-bit Address Mode with "mod r/m" Byte

mod r/m	Effective Address
00 000	DS:[BX + SI]
00 001	DS:[BX + DI]
00 010	SS:[BP + SI]
00 011	SS:[BP + DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BP + DI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by r/m During 16-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m During 32-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by reg or r/m during 16-Bit Data Operations:		
mod r/m	Function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by reg or r/m during 32-Bit Data Operations:		
mod r/m	Function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode ("mod r/m" byte and "s-i-b" byte present):

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

NOTE:
Mod field in "mod r/m" byte; ss, index, base fields in "s-i-b" byte.

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg**
101	EBP
110	ESI
111	EDI

****IMPORTANT NOTE:**
When index field is 100, indicating "no index register," then ss field MUST equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

10.2.3.5 Encoding of Operation Direction (d) Field

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory <- - Register "reg" Field Indicates Source Operand; "mod r/m" or "mod ss index base" Indicates Destination Operand
1	Register <- - Register/Memory "reg" Field Indicates Destination Operand; "mod r/m" or "mod ss index base" Indicates Source Operand

10.2.3.6 Encoding of Sign-Extend (s) Field

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

s	Effect on Immediate Data8	Effect on Immediate Data 16|32
0	None	None
1	Sign-Extend Data8 to Fill 16-Bit or 32-Bit Destination	None

10.2.3.7 Encoding of Conditional Test (tttn) Field

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with n indicating to use the condition (n = 0) or its negation (n = 1), and ttt giving the condition to test.

Mnemonic	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

10.2.3.8 Encoding of Control or Debug or Test Register (eee) Field

For the loading and storing of the Control, Debug and Test registers.

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	CR2
011	CR3
Do not use any other encoding	

When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding	

When Interpreted as Test Register Field

eee Code	Reg Name
011	TR3
100	TR4
101	TR5
110	TR6
111	TR7
Do not use any other encoding	

	Instruction									Optional Fields	
	First Byte				Second Byte						
1	11011	OPA		1	mod		1	OPB	r/m	s-i-b	disp
2	11011	MF		OPA	mod		OPB		r/m	s-i-b	disp
3	11011	d	P	OPA	1	1	OPB		ST(i)		
4	11011	0	0	1	1	1	1	OP			
5	11011	0	1	1	1	1	1	OP			
	15–11	10	9	8	7	6	5	4 3	2 1 0		

10.2.4 ENCODING OF FLOATING POINT INSTRUCTION FIELDS

Instructions for the FPU assume one of the five forms shown in the following table. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011B.

OP = Instruction opcode, possible split into two fields OPA and OPB

MF = Memory Format
- 00—32-bit real
- 01—32-bit integer
- 10—64-bit real
- 11—16-bit integer

P = Pop
- 0—Do not pop stack
- 1—Pop stack after operation

d = Destination
- 0—Destination is ST(0)
- 1—Destination is ST(i)

R XOR d = 0—Destination (op) Source
R XOR d = 1—Source (op) Destination

ST(i) = Register stack element *i*
- 000 = Stack top
- 001 = Second stack element
- •
- •
- •
- 111 = Eighth stack element

mod (Mode field) and r/m (Register/Memory specifier) have the same interpretation as the corresponding fields of the integer instructions.

s-i-b (Scale Index Base) byte and disp (displacement) are optionally present in instructions that have mod and r/m fields. Their presence depends on the values of mod and r/m, as for integer instructions.

11.0 DIFFERENCES BETWEEN THE Intel486™ MICROPROCESSOR AND THE 386 MICROPROCESSOR PLUS THE 387 MATH COPROCESSOR EXTENSION

The differences between the Intel486 Microprocessor and the 386 Microprocessor are due to performance enhancements. The differences between the microprocessors are listed below.

1. Instruction clock counts have been reduced to achieve higher performance. See Section 10.
2. The Intel486 Microprocessor bus is significantly faster than the 386 Microprocessor bus. Differences include a 1X clock, parity support, burst cycles, cacheable cycles, cache invalidate cycles and 8-bit bus support. The Hardware Interface and Bus Operation Sections (Sections 6 and 7) of the data sheet should be carefully read to understand the Intel486 Microprocessor bus functionality.
3. To support the on-chip cache new bits have been added to control register 0 (CD and NW) (Section 2.1.2.1), new pins have been added to the bus (Section 6) and new bus cycle types have been added (Section 7). The on-chip cache needs to be enabled after reset by clearing the CD and NW bit in CR0.
4. The complete 387 math coprocessor instruction set and register set have been added. No I/O cycles are performed during Floating Point instructions. The instruction and data pointers are set to 0 after FINIT/FSAVE. Interrupt 9 can no longer occur, interrupt 13 occurs instead.
5. The Intel486 Microprocessor supports new floating point error reporting modes to guarantee DOS compatibility. These new modes required a new bit in control register 0 (NE) (Section 2.1.2.1) and new pins (FERR# and IGNNE#) (Section 6.2.13 and 7.2.14).
6. In some cases FERR# is asserted when the next floating point instruction is encountered and in other cases it is asserted before the next floating point instruction is encountered, depending upon the execution state the instruction causing exception (see Sections 6.2.13 and 7.2.14). For both of these cases, the 387 Math Coprocessor asserts ERROR# when the error occurs and does not wait for the next floating point instruction to be encountered.
7. Six new instructions have been added:
   Byte Swap (BSWAP)
   Exchange-and-Add (XADD)
   Compare and Exchange (CMPXCHG)
   Invalidate Data Cache (INVD)
   Write-back and Invalidate Data Cache (WBINVD)
   Invalidate TLB Entry (INVLPG)
8. There are two new bits defined in control register 3, the page table entries and page directory entries (PCD and PWT) (Section 4.5.2.5).
9. A new page protection feature has been added. This feature required a new bit in control register 0 (WP) (Section 2.1.2.1 and 4.5.3).
10. A new Alignment Check feature has been added. This feature required a new bit in the flags register (AC) (Section 2.1.1.3) and a new bit in control register 0 (AM) (Section 2.1.2.1).
11. The replacement algorithm for the translation lookaside buffer has been changed from a random algorithm to a pseudo least recently used algorithm like that used by the on-chip cache. See Section 5.5 for a description of the algorithm.
12. Three new testability registers, TR3, TR4 and TR5, have been added for testing the on-chip cache. TLB testability has been enhanced. See Section 8.
13. The prefetch queue has been increased from 16 bytes to 32 bytes. A jump always needs to execute after modifying code to guarantee correct execution of the new instruction.
14. After reset, the ID in the upper byte of the DX register is 04. The contents of the base registers including the floating point registers may be different after reset.

12.0 OVERDRIVE PROCESSOR SOCKET

Inclusion of the OverDrive Processor Socket in systems based on Intel486 DX Microprocessors provides the end-user with an easy and cost-effective way to increase system performance. The paradigm of simply installing an additional component into an empty OverDrive Processor Socket to achieve enhanced system performance is familiar to the millions of end-users and dealers who have purchased Intel Math CoProcessor upgrades to boost system floating point performance. The OverDrive Processor provides an overall performance increase for systems based on Intel486 DX Microprocessors.

As a new system architectural feature, the provision of the OverDrive Processor Socket as a means for PC users to take advantage of the ever more rapid advances in software and hardware technology will help to maintain the competitiveness of X86 PC-compatible systems over other architectures into the future.

The majority of upgrade installations which take advantage of the OverDrive Processor Socket will be performed by end-users and resellers. Therefore, it is important that the design be "end-user easy", and that the amount of training and technical expertise required to install the OverDrive Processor be minimized. Upgrade installation instructions should be clearly described in the system user's manual. In addition, by making installation simple and foolproof, PC manufacturers can reduce the risk of system damage, warranty claims and service calls. Feedback from Intel's Math CoProcessor customers highlights three main characteristics of end-user easy designs: accessible OverDrive Processor Socket location, clear indication of component orientation, and minimization of insertion force.

OverDrive Processor Socket Location: The OverDrive Processor Socket for Intel486 DX and Intel486 SX Microprocessor based systems is an empty socket which can be located on either the motherboard or modular CPU card. The OverDrive Processor Socket should be easily accessible for installation and readily visible when the PC case is removed. The OverDrive Processor Socket should not be located in a position that requires removal of any other hardware (such as hard disk drives) in order to install the OverDrive Processor. Since Math CoProcessor sockets are typically found near the CPU socket on the motherboard, similarly locating the OverDrive Processor Socket near the CPU further adds to the ease of installation.

Component Orientation: The most common mistake made by end-users and resellers when installing Math CoProcessor upgrades is incorrect orientation of the chip. This can result in irreversible damage to the chip and/or the PC. To solve this problem, Intel has designed the OverDrive Processor with a 169 pin Pin Grid Array (PGA) pinout, with the 169th pin as a non-electrical "key pin" used to ensure proper orientation of the OverDrive Processor by the PC user. The OverDrive Processor Socket should, therefore, be a 169 pin PGA socket compatible with the OverDrive Processor pinout.(1) In addition, the location of the key pin should be clearly marked on the motherboard or CPU card, for example by silk screening.

Insertion Force: The third major concern voiced by end-users refers to how much pressure should be exerted on the chip and PC board for proper installation without damage. This becomes even more of a concern with the larger 169 pin components which require up to 150 pounds of pressure for insertion into a standard screw machine socket. This level of pressure can easily result in cracked traces and stress to solder joints. To minimize the risk of system damage, it is recommended that a Zero Insertion Force (ZIF) socket be used for the OverDrive Processor Socket. Designing with a ZIF socket eliminates the need to design in additional structural support to prevent flexing of the PC board during installation, and results in improved end-user and reseller product satisfaction due to easy "drop-in" installation.

12.1 OverDrive Processor Overview

The Intel OverDrive Processor is essentially an enhanced Intel486 Microprocessor. There are three functional differences between the Intel OverDrive Processor and Intel486 Microprocessors. First, the Intel OverDrive Processor has an internal clock doubling circuit which decreases the time required to execute instructions. Second, the Intel OverDrive Processor does not support the JTAG boundary scan test feature (available with the PQFP version of the Intel486 DX Microprocessor). Third, the Intel OverDrive Processor has a different CPU revision identification than the Intel486 DX CPU. These three differences are described in the following sections according to how they effect the CPU functionality.

12.1.1 HARDWARE INTERFACE

The Intel OverDrive Processor bus has been designed to be identical with the Intel486 Microprocessor bus. Although the external clock is internally doubled and data and instructions are manipulated in the CPU core at twice the external frequency, the external bus is functionally identical with the Intel486 CPU.

The four boundary scan test signals (TCK, Test clock; TMS, Test Mode select; TDI, Test Data Input; TDO, Test Data Output), defined for the PQFP Intel 486 SX CPU, are not specified for the Intel OverDrive Processor.

The UP# (Upgrade Present) signal, which is defined as an input for the PQFP Intel486 CPU, is an output signal on the Intel OverDrive Processor. The UP# pin on the Intel OverDrive Processor provides a logical low output signal which can be used to enable logic to recognize and configure the system for the Intel OverDrive Processor.

The DX register always contains the component identifier at the conclusion of RESET. The Intel OverDrive Processor has a different revision identifier in the DL register than the Intel486 DX Microprocessor. When the OverDrive Processor is installed in a system the component identifier is supplied by the OverDrive Processor, rather than the original CPU. The stepping identification portion of the component identification will change with different revisions of the OverDrive Processor. The designer should only assume that the component identification for the OverDrive Processor will be 043xH, where 'x' is the stepping identifier.

12.1.2 TESTABILITY

As detailed in Section 13.1.1, the Intel OverDrive Processor does not support the JTAG boundary scan testability feature.

12.1.3 INSTRUCTION SET SUMMARY

The Intel OverDrive Processor supports all Intel486 extensions to the 8086/80186/80286 instruction set. In general, instructions will execute faster on the Intel OverDrive Processor than the Intel486 Microprocessor. Specifically, an instruction that only uses memory from the on-chip cache executes at the full core clock rate while all bus accesses execute at the bus clock rate. To calculate the elapsed time of an instruction, the number of clock counts for that instruction must be multiplied by the clock period for the system. The instruction set clock count summary tables from Section 10.0 can be used for the OverDrive Processor witth the following modifications:

— Clock counts for a cache hit: This value represents the number of internal CPU core clocks for an instruction that requires no external bus accesses or the base core clocks for an instruction requiring external bus accesses.

— Penalty clock counts for a cache miss: This value represents the worst-case approximation of the additional number of external clock counts that are required for an instruction which must access the external bus for data (a cache miss). This number must be multiplied by 2 to convert it to an equal number of internal CPU core clock counts and added to the base core clocks to compute the total number of core clocks for this instruction.

The actual number of core clocks for an instruction with a cache miss may be less than the base clock counts (from the cache hit column) plus the penalty clock counts (2 times the cache miss column number). The clock counts in the cache miss penalty column can be a cumulative value of external bus clocks (for data reads) and internal clocks for manipulating the data which has been loaded from the external bus. The number of clocks which are related to external bus accesses are correctly represented in terms of internal core clocks by multiplying by two. However, the clock counts related to internal data manipulation should not be multiplied by two. Therefore the total number of CPU core clock counts for an instruction with a cache miss represents a worst-case approximation.

To calculate the execution time for an OverDrive Processor instruction, multiply the total CPU core clock counts by the core clock period. For example, in a 25 MHz system the core clock period is 50 ns (1/50 MHz).

Additionally, the assumptions specified below should be understood in order to estimate instruction execution time.

A cache miss will force the OverDrive Processor to run an external bus cycle. The Intel486 DX microprocessor 32-bit burst bus is defined as r−b−w.

Where:

r = The number of bus clocks in the first cycle of a burst read or the number of clocks per data cycle is a non-burst read.

b = The number of bus clocks for the second and subsequent cycles in a burst read.

w = The number of bus clocks for a write.

The fastest bus the OverDrive Processor can support is 2−1−2 assuming 0 wait states. The clock counts in the cache miss penalty column assume a 2−1−2 bus. For slower busses add r−2 clocks to the cache miss penalty for the first dword accessed. Other factors also affect instruction clock counts.

Instruction Clock Count Assumptions

1. The external bus is available for reads or writes at all times. Else add bus clocks to reads until the bus is available
2. Accesses are aligned. Add three core clocks to each misaligned access.

3. Cache fills complete before subsequent accesses to the same line. If a read misses the cache during a cache fill due to a previous read or prefetch, the read must wait for the cache fill to complete. If a read or write accesses a cache line still being filled, it must wait for the fill to complete.
4. If an effective address is calculated, the base register is not the distination register of the preceding instruction. If the base register is the destination register of the preceding instruction add 1 to the core clock counts shown. Back-to-back PUSH and POP instructions are not affected by this rule.
5. An effective address calculation uses one base register and does not use an index register. However, if the effective address calculation uses an index register. 1 core clock may be added to the clock shown.
6. The target of a jump is in the cache. If not, add r clocks for accessing the destination instruction of a jump. If the destination instruction is not completely contained in the first dword read, add a maximum of 3b bus clocks. If the destination instruction is not completely contained in the first 16 byte burst, add a maximum of another r+3b bus clocks.
7. If no write buffer delay, w bus clocks are added only in the case in which all write buffers are full.
8. Displacement and immediate not used together. If displacement and immediate used together, 1 core clock may be added to the core clock count shown.
9. No invalidate cycles. Add a delay of 1 bus clock for each invalidate cycle if the invalidate cycle contends for the internal cache/external bus when the OverDrive Processor needs to use it.
10. Page translation hits in TLB. A TLB miss will add 13, 21 or 28 bus clocks + 1 possible core clock to the instruction depending on whether the Accessed and/or Dirty bit in neither, one or both of the page entries needs to be set in memory. This assumes that neither page entry is in the data cache and a page fault does not occur on the address translation.
11. No exceptions are detected during instruction execution. Refer to interrupt core Clock Counts Table for extra clocks if an interrupt is detected.
12. Instructions that read multiple consecutive data items (i.e., task switch, POPA, etc.) and miss the cache are assumed to start the first access on a 16-byte boundary. If not, an extra cache line fill may be necessary which may add up to (r+3b) bus clocks to the cache miss penalty.

12.2 Intel OverDrive™ Processor Circuit Design

Figure 12.1 shows the interface circuit for the Intel486 DX CPU and the OverDrive Processor socket. This circuit allows Intel486 DX CPU-based systems to be upgraded with the OverDrive Processor.

12.2.1 UPGRADE CIRCUIT FOR PGA INTEL486 DX BASED SYSTEMS

The Intel OverDrive Processor Socket Circuit for Intel486 DX CPU based systems allows the Intel486 DX CPU complete control of the system when the Intel OverDrive Processor Socket is unpopulated. The HLDA signal from the Intel OverDrive Processor Socket should be tied low through a resistor while the UP# and FERR# signals from the Intel OverDrive Processor Socket should be tied high through a resistor to insure that the Intel486 DX CPU functions correctly when an Intel OverDrive Processor Socket component is not installed.

When the Intel OverDrive Processor is installed, the Upgrade Present output, UP# pin, causes the FLUSH# and BOFF# signals to be driven active to the Intel486 DX CPU. When the Intel486 DX CPU samples FLUSH# active during reset, the Intel486 DX CPU enters tri-state output test mode after reset, which causes the Intel486 DX CPU to float all of its output signals. To float most of the Intel486 DX CPU's output pins before the end of reset, BOFF# is also driven active to the Intel486 DX CPU. BOFF# immediately causes all output signals to float except PCHK#, BREQ, HLDA and FERR#.

In addition to floating the Intel486 DX CPU's outputs, the Intel486 DX CPU's HLDA and FERR# signals must be gated to prevent potential bus contention with the Intel OverDrive Processor's HLDA and FERR# signals during reset. During reset the Intel486 DX CPU may not recognize HOLD active because BOFF# is driven active to the Intel486 DX CPU by the Intel OverDrive Processor. If the Intel486 DX CPU does not recognize HOLD active, it will not drive HLDA active. However, the Intel OverDrive Processor will recognize HOLD active and drive HLDA. By gating the HLDA signals from the Intel486 DX CPU and Intel OverDrive Processor Socket, bus contention is avoided if HOLD is driven active during reset. Because the state of FERR# is undefined during reset, bus contention is also avoided by gating FERR#.

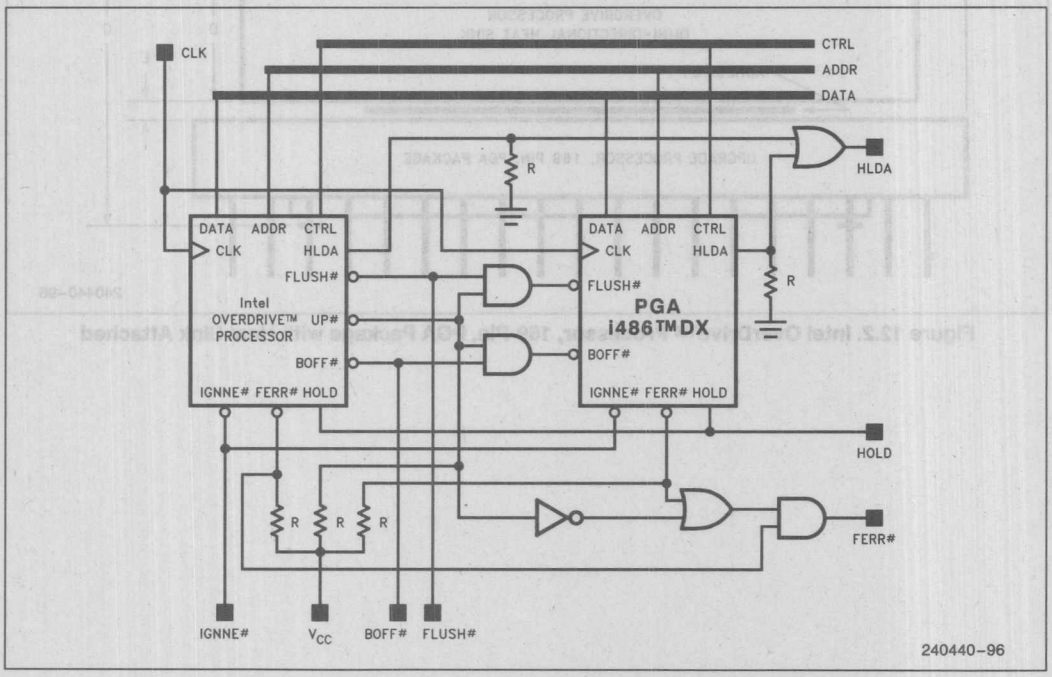


Figure 12.1. Intel OverDrive™ Socket Circuit Diagram for PGA Intel486™ DX CPU Based Systems

12.3 Socket Layout

This section discusses three aspects for the OverDrive Processor Socket: size, upgradability, and vendors.

12.3.1 PHYSICAL DIMENSIONS

The OverDrive Processor Socket for Intel486 DX microprocessor-based systems is equivalent to a standard 169-lead PGA package.

The OverDrive Processor will be provided with a heat sink attached (see Figure 12-2), to dissipate heat.

The maximum and minimum dimensions of the OverDrive Processor package with the heat sink are shown in Table 12-1.

Table 12.1. OverDrive Processor, 169-Pin, PGA Package Dimensions with Heat Sink Attached

Dimension (Inches)	Minimum	Maximum
A. Heat Sink Width	1.520	1.550
B. PGA Package Width	1.735	1.765
C. Heat Sink Edge Gap	0.065	0.155
D. Heat Sink Height	0.212	0.260
E. Adhesive Thickness	0.008	0.012
F. Package Height from Stand-Offs	0.140	0.180
G. Total Height from Stand-Offs to Top of Heat Sink	0.360	0.452

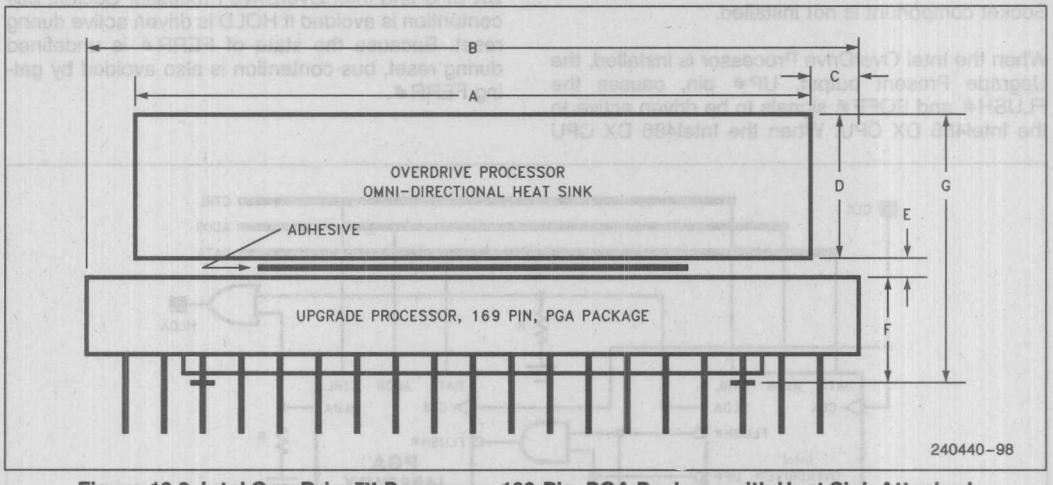
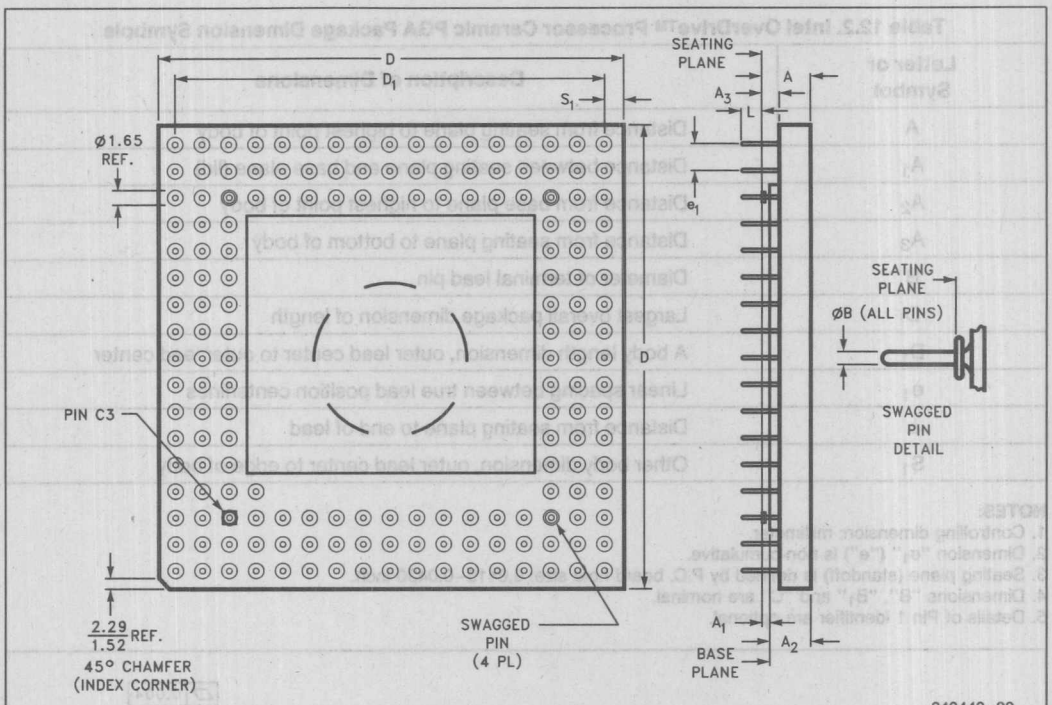


Figure 12.2. Intel OverDrive™ Processor, 169-Pin, PGA Package with Heat Sink Attached



Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		0.140	0.180	
A_1	0.64	1.14	SOLID LID	0.025	0.045	SOLID LID
A_2	2.8	3.5	SOLID LID	0.110	0.140	SOLID LID
A_3	1.14	1.40		0.045	0.055	
B	0.43	0.51		0.017	0.020	
D	44.07	44.83		1.735	1.765	
D_1	40.51	40.77		1.595	1.605	
e_1	2.29	2.79		0.090	0.110	
L	2.54	3.30		0.100	0.130	
N	169			169		
S_1	1.52	2.54		0.060	0.100	
ISSUE	IWS REV X 7/15/88					

Figure 12.3. Intel OverDrive™ Processor, 169-Lead Ceramic PGA Package Dimensions

Table 12.2. Intel OverDrive™ Processor Ceramic PGA Package Dimension Symbols

Letter or Symbol	Description of Dimensions
A	Distance from seating plane to highest point of body
A_1	Distance between seating plane and base plane (lid)
A_2	Distance from base plane to highest point of body
A_3	Distance from seating plane to bottom of body
B	Diameter of terminal lead pin
D	Largest overall package dimension of length
D_1	A body length dimension, outer lead center to outer lead center
e_1	Linear spacing between true lead position centerlines
L	Distance from seating plane to end of lead
S_1	Other body dimension, outer lead center to edge of body

NOTES:
1. Controlling dimension: millimeter.
2. Dimension "e_1" ("e") is non-cumulative.
3. Seating plane (standoff) is defined by P.C. board hole size: 0.0415–0.0430 inch.
4. Dimensions "B", "B_1" and "C" are nominal.
5. Details of Pin 1 identifier are optional.

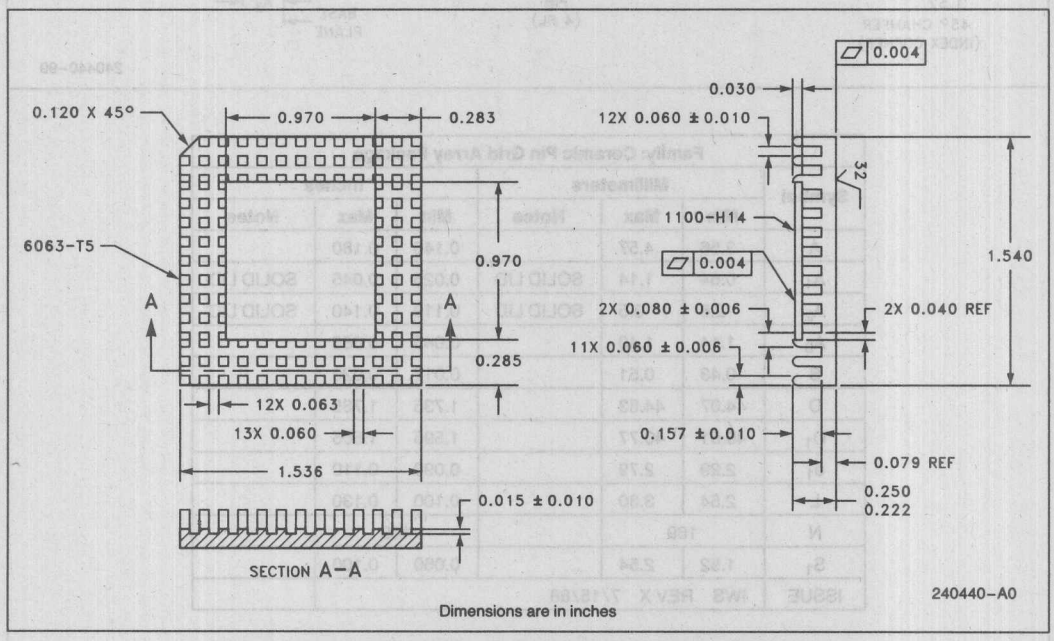


Figure 12.4. Intel OverDrive™ Processor Heat Sink Dimensions

12.3.2 "END USER EASY" UPGRADABILITY

PC buyers value easy and safe upgrade installation. PC manufacturers can make the Intel OverDrive Processor installation in the Intel OverDrive Processor Socket simple and foolproof for the end user and reseller by implementing the suggestions listed in Table 12-3.

Table 12.3. Socket and Layout Considerations

"End User Easy" Feature	Implementation
Visible OverDrive Processor Socket	The Intel OverDrive Processor Socket should be easily visible when the PC's cover is removed. Label the Intel OverDrive Processor Socket and the location of pin 1 by silk screening this information on the PC board.
Accessible Overdrive Processor Socket	Make the Intel OverDrive Processor Socket easily accessible to the end user (i.e., do not place the Intel OverDrive Processor Socket under a disk drive). If a Low Insertion Force (LIF) or screw machine socket is used, position the Intel OverDrive Processor Socket on the PC board such that there is ample clearance around the socket.
Foolproof Chip Orientation	Intel packages all Intel OverDrive Processors in a 169-pin, PGA package. The 169th pin is called the "key" pin and insures that the Intel OverDrive Processor fits into a 169-pin socket in only the correct orientation. Supplying a 169-pin socket as the Intel OverDrive Processor Socket eliminates the possibility of end users or resellers damaging the PC board or Intel OverDrive Processor by powering up the system with the Intel OverDrive Processor incorrectly oriented.
Zero Insertion Force Upgrade Socket	The high pin count of the Intel OverDrive Processor makes the insertion force required for installation in a screw machine PGA socket excessive for end users or resellers. Even most Low Insertion Force (LIF) sockets often require more than 60 lbs. of insertion force. A Zero Insertion Force (ZIF) socket insures that the chip insertion force does not damage the PC board. If the ZIF socket has a handle, be sure to allow enough clearance for the socket handle. If a LIF or screw machine socket is used, additional PC board support is recommended.
"Plug and Play"	Jumper or switch changes should not be needed to electrically configure the system for the Intel OverDrive Processor.
Thorough Documentation	Describe the Intel OverDrive Processor's installation procedure in the PC's User's Manual.

12.3.3 ZIF and LIF SOCKET VENDORS

The following lists provide examples of sockets which can be used as the Intel OverDrive Socket for Intel486 DX CPU based systems.

NOTE:
This is not a comprehensive list. Intel has not tested the sockets listed below and cannot guarantee that these sockets will meet every PC manufacturer's specific requirements.

Zero Insertion Force Upgrade Sockets and Vendors:

1. AMP Inc.
   P.O. Box 3608
   Harrisburg, PA 17105-3608
   Tel: (800) 522-6752
   Part Number: 55287-3
   Contact: Rick Simonic, New Product Manager
   (717) 561-6143

2. Aries Electronics
   P.O. Box 130
   Frenchtown, NJ 08825
   Tel: (908) 996-6841
   Part Number: 169-PRS17012-10
   Contact: Frank Folmsbee, Marketing Manager
   (908) 996-6841

3. JAE
   599 N. Mathilda Ave., Suite 8
   Sunnyvale, CA 94086
   Tel: (408) 733-0493
   Part Number: PCPS-169-002
   Contact: Bob Gerleman, Western Sales Manager
   (408) 733-0493

4. Thomas and Betts
   200 Executive Center Drive
   P.O. Box 24901
   Greenville, SC 29616-2401
   Tel: (803) 676-2900
   Part Number: PGA169A17-S-1AC
   Contact: Scott Roland,
   Product Marketing Manager
   (803) 676-2910

5. Yamaichi Electronics
   1420 Koll Circle, Suite B
   San Jose, CA 95112
   Tel: (408) 452-0797
   Part Number: NP111-16911-G4
   Contact: Jim Bennett, Sales Manager
   (408) 452-0797

Low Insertion Force Sockets and Vendors:

1. AMP Inc.
   P.O. Box 3608
   Harrisburg, PA 17105-3608
   Tel: (800) 522-6752
   Part Number:
   (Premium Base Material) 55589-5
   (Standard Base Material) 916227-3

2. Thomas and Betts
   200 Executive Center Drive
   P.O. Box 24901
   Greenville, SC 29616-2401
   Tel: (803) 676-2900
   Part Number: LPG169A17-S-1AC

12.4 Thermal Management

The OverDrive Processor Socket must be designed to dissipate the heat generated by the OverDrive Processor. In the following Sections the airflow required over the OverDrive Processor Socket is calculated for a hypothetical system design.

12.4.1 THERMAL CALCULATIONS FOR HYPOTHETICAL SYSTEM

The maximum temperature specification for the OverDrive Processor is 85°C (with heat sink attached). Therefore, the temperature of the heat sink surface (T_S) cannot exceed 85°C under the worst case specified operating conditions for the system. The variables which affect the heat sink temperature include ambient temperature inside the system box (T_A), V_{CC}, and I_{CC}. An equation for the approximate OverDrive Processor temperature (T_S) is:

$$T_S = T_A + \text{Power} * \theta_{SA} \qquad \text{where Power} = V_{CC} * I_{CC}$$

In the above equation, the variables under worst case conditions are specified as follows:

T_S: Specified as 85°C for the OverDrive Processor (See Figure 12-5).

T_A: Specified by the PC manufacturer for the worst case system operating conditions.

V_{CC}: Specified for the OverDrive Processor as 5V.

I_{CC}: Specified for the OverDrive Processor and related to clock frequency.

θ_{SA}: $\theta_{SA} = \theta_{JA} - \theta_{JS}$.
θ_{JA} and θ_{JS} are specified in Table 13-4.

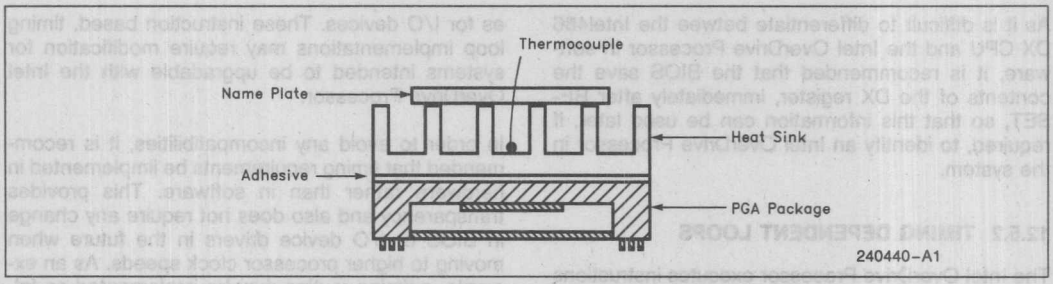


Figure 12.5. Heat Sink Measurement (0.005" Dia. Thermocouple) on the Center of Heat Sink with a 90° Angle Adhesive Bond Through a Hole Drilled Through the Center of the Name Plate.

The OverDrive Processor for Intel486 DX CPU-based systems will be provided with a heat sink. The θ_{JS} and θ_{JA} values for the OverDrive Processor with a heat sink are shown in Table 12-4. The maximum T_A values for the 25 MHz and 33 MHz OverDrive Processor are shown in Table 12-5. The maximum T_A values shown in Table 12-5 were calculated using T_S = 85°C, V_{CC} = 5V, the maximum I_{CC} values, and the θ_{JA} and θ_{JS} values shown in Table 12-4.

Table 12.4. Thermal Resistance (°C/W) θ_{JS} and θ_{JA}

OverDrive with Heat Sink	θ_{JS}	Airflow (ft/min, LFM)				
	2.5°C/W	0*	200	400	600	800
θ_{JA} (°C/W)		14.0	10.0	7.5	6.2	5.7

NOTE:
*The thermal resistance from the junction to ambient (θ_{JA}) in static air is actually a linear function of power dissipation. The value shown in the table (14.0 °C/W) represents the worst case expected value.

Table 12.5. Maximum T_A for 25 MHz and 33 MHz OverDrive Processor

OverDrive Processor with Heat Sink		Linear Airflow (ft/min)				
	f_{CLK} (MHz)	0	200	400	600	800
T_A (°C)	25	30	49	61	67	70
	33	16	40	55	63	66

12.4.2 HEAT SINKS

The OverDrive Processor is shipped with a heat sink attached. Because of the heat sink, it is vital that vertical clearance is provided for the OverDrive Processor Socket. The height of the package and the heat sink is shown in Table 12-1 in Section 12.2.1.

12.5 BIOS and Software

The following should be considered when designing the Upgrade Socket for a Intel486 DX2 microprocessor-based system.

12.5.1 INTEL OVERDRIVE PROCESSOR DETECTION

The component identifier and stepping/revision identifier for the Intel OverDrive Processor is readable in the DH and DL registers respectively, immediately after RESET, where

DH = 15h

DL = 30h–3Fh

As it is difficult to differentiate betwee the Intel486 DX CPU and the Intel OverDrive Processor in software, it is recommended that the BIOS save the contents of the DX register, immediately after RESET, so that this information can be used later, if required, to identify an Intel OverDrive Processor in the system.

12.5.2 TIMING DEPENDENT LOOPS

The Intel OverDrive Processor executes instructions at twice the frequency of the input clock. Thus software (or instuction based) timing loops will execute faster on the Intel OverDrive Processor than on the Intel486 DX or Intel486 SX CPU (at the same input clock frequency). Instructions such as NOP, LOOP, and jMP $ + 2, have been used by BiOS to implement timing loops that are required, for example, to enforce recovery time between consecutive accesses for I/O devices. These instruction based, timing loop implementations may require modification for systems intended to be upgradable with the Intel OverDrive Processor.

In order to avoid any incompatibilities, it is recommended that timing requirements be iimplemented in hardware rather than in software. This provides transparency and also does not require any change in BIOS or I/O device drivers in the future when moving to higher processor clock speeds. As an example, a timing routine may be implemented as follows: The software performs a dummy I/O instruction to an unused I/O port. The hardware for the bus controller logic recognizes this I/O instruction and delays the termination of the I/O cycle to the CPU by keeping RDY# or BRDY# deasserted for the appropriate amount of time.

12.6 OverDrive Processor Socket Pinout

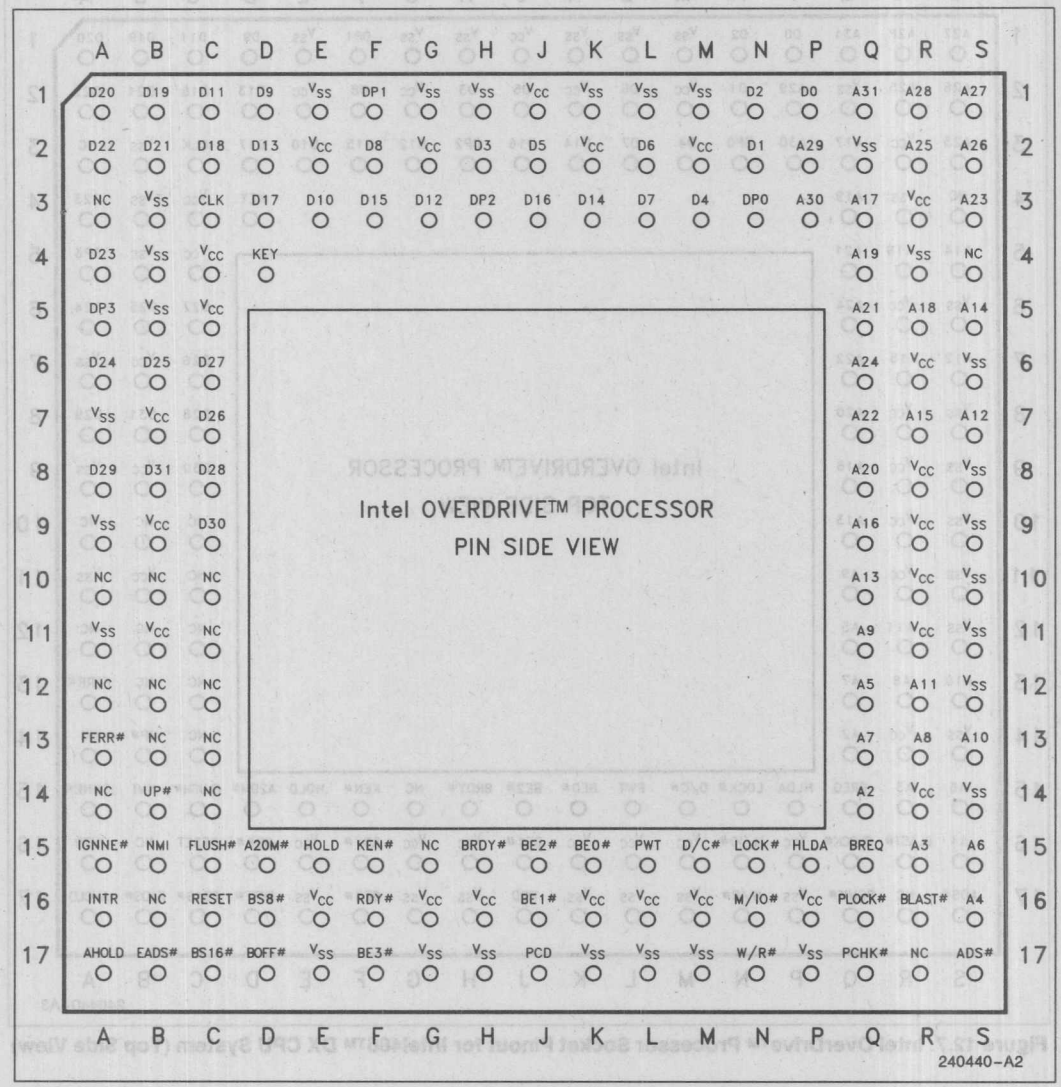


Figure 12.6 Intel OverDrive™ Processor Socket Pinout for Intel486™ DX CPU System (Pin Side View)

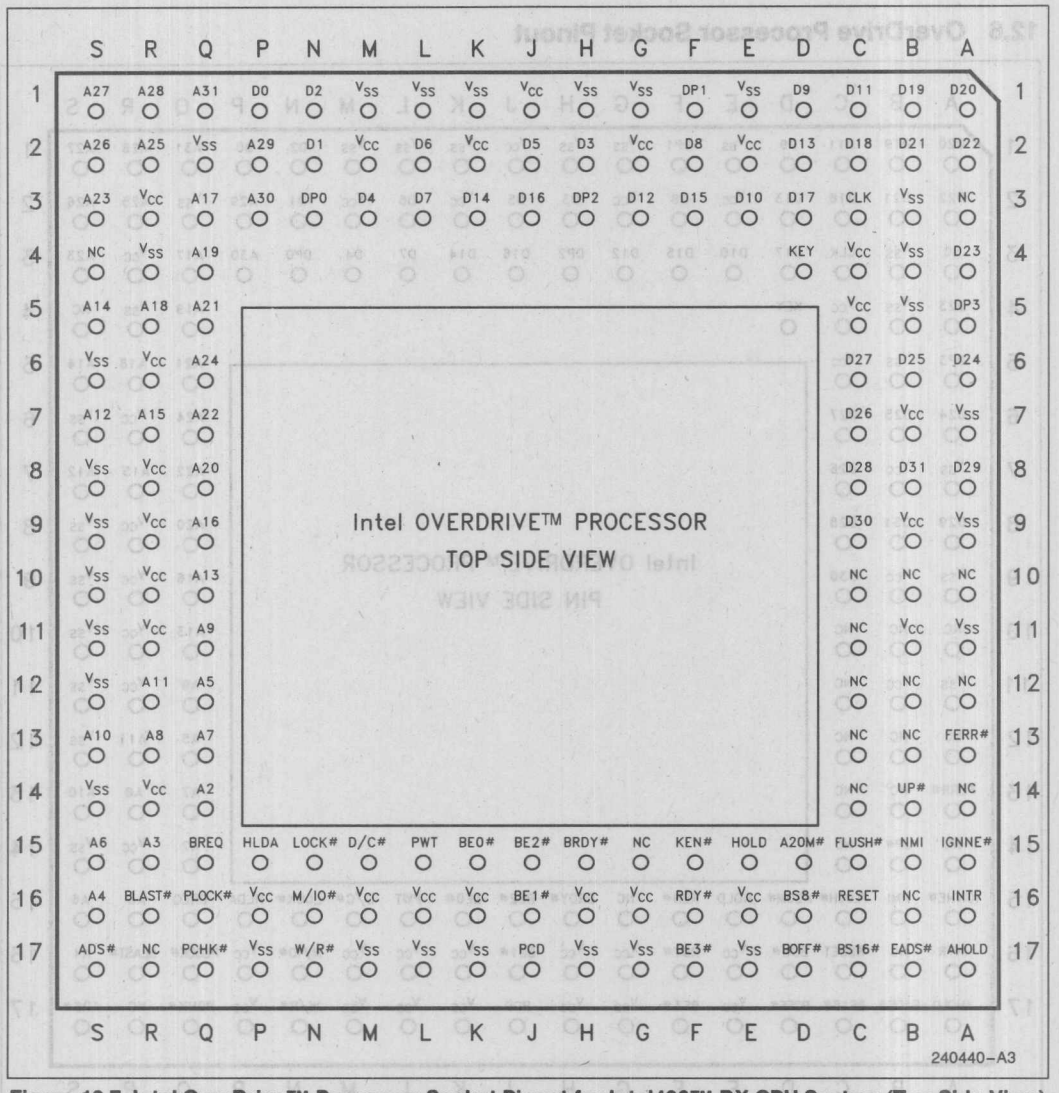


Figure 12.7. Intel OverDrive™ Processor Socket Pinout for Intel486™ DX CPU System (Top Side View)

Table 12.6. Pin Cross Reference by Pin Name

Address		Data		Control		N/C	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	A10	B7	A7
A_3	R15	D_1	N2	ADS#	S17	A12	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	A14	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	B12	C4	B3
A_6	S15	D_4	M3	BE1#	J16	B13	C5	B4
A_7	Q13	D_5	J2	BE2#	J15	C10	E2	B5
A_8	R13	D_6	L2	BE3#	F17	C13	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16	G15	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17	R17	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15	S4	H16	G17
A_{12}	S7	D_{10}	E3	BREQ#	Q15	A3	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16	B10	K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17	B16	K16	K1
A_{15}	R7	D_{13}	D2	CLK	C3	C11	L16	K17
A_{16}	Q9	D_{14}	K3	D/C#	M15	C12	M2	L1
A_{17}	Q3	D_{15}	F3	DP0	N3	C14	M16	L17
A_{18}	R5	D_{16}	J3	DP1	F1		P16	M1
A_{19}	Q4	D_{17}	D3	DP2	H3		R3	M17
A_{20}	Q8	D_{18}	C2	DP3	A5		R6	P17
A_{21}	Q5	D_{19}	B1	EADS#	B17		R8	Q2
A_{22}	Q7	D_{20}	A1	FERR#	A13		R9	R4
A_{23}	S3	D_{21}	B2	FLUSH#	C15		R10	S6
A_{24}	Q6	D_{22}	A2	HLDA	P15		R11	S8
A_{25}	R2	D_{23}	A4	HOLD	E15		R14	S9
A_{26}	S2	D_{24}	A6	IGNNE#	A15			S10
A_{27}	S1	D_{25}	B6	INTR	A16			S11
A_{28}	R1	D_{26}	C7	KEN#	F15			S12
A_{29}	P2	D_{27}	C6	LOCK#	N15			S14
A_{30}	P3	D_{28}	C8	M/IO#	N16			
A_{31}	Q1	D_{29}	A8	NMI	B15			
		D_{30}	C9	PCD	J17			
		D_{31}	B8	PCHK#	Q17			
				PWT	L15			
				PLOCK#	Q16			
				RDY#	F16			
				RESET	C16			
				UP#	B14			
				W/R#	N17			
				KEY	D4			

Table 12-7. Intel OverDrive™ Processor Socket Pin Description

Symbol	Type	Name and Function
Intel486 DX2 CPU INTERFACE		
UP#	O	The *Upgrade Present* pin is used to signal the Intel486 DX microprocessor to float its outputs and get-off the bus. It is active low and is never floated. UP# is driven low at power-up and remains active for the entire duration of the Upgrade Processor operation.
KEY PIN		
KEY		The Key pin is an electrically non-functional pin which is used to ensure correct orientation for 169-pin upgrade products.

12.7 D.C./A.C. Specifications

The electrical specifications in this section represent the electrical interface of the Upgrade Processor for a Intel486 DX microprocessor-based system. The OverDrive Processor is compatible to the maximum ratings and A.C. Specifications of the Intel486 DX Microprocessor. Table 12-8 provides the D.C. Operating Conditions for the OverDrive Processor.

Table 12-8. Intel OverDrive™ Processor Socket D.C. Parametric Values(1)

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 2)
V_{OH}	Output High Voltage	2.4		V	(Note 3)
I_{CC}	Power Supply Current CLK = 25 MHz CLK = 33 MHz		 950 1200	mA	(Note 4)
I_{LI}	Input Leakage Current		±15	μA	(Note 5)
I_{IH}	Input Leakage Current		200	μA	(Note 6)
I_{IL}	Input Leakage Current		−400	μA	(Note 7)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		13	pF	F_C = 1 MHz(8)
C_O	I/O or Output Capacitance		17	pF	F_C = 1 MHz(8)
C_{CLK}	CLK Capacitance		15	pF	F_C = 1 MHz(8)

NOTES:
1. Functional operating range: V_{CC} = 5V; T_S = 0°C to +85°C.
2. This parameter is measured at:
   — Address, Data, BEn 4.0 mA
   — Definition, Control 5.0 mA
3. This parameter is measured at:
   — Address, Data, BEn −1.0 mA
   — Definition, Control −0.9 mA
4. Typical supply current:
   775 mA @ CLK = 25 MHz
   975 mA @ CLK = 33 MHz
5. This parameter is for inputs without pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
6. This parameter is for inputs with pulldowns and V_{IH} = 2.4V.
7. This parameter is for inputs with pullups and V_{IL} = 0.45V.
8. Not 100% tested.

13.0 ELECTRICAL DATA

The following sections describe recommended electrical connections for the Intel486 Microprocessor, and its electrical specifications.

13.1 Power and Grounding

13.1.1 POWER CONNECTIONS

The Intel486 Microprocessor is implemented in CHMOS IV technology and has modest power requirements. However, its high clock frequency output buffers can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 24 V_{CC} and 28 V_{SS} pins feed the Intel486 Microprocessor.

Power and ground connections must be made to all external V_{CC} and GND pins of the Intel486 Microprocessor. On the circuit board, all V_{CC} pins must be connected on a V_{CC} plane. All V_{SS} pins must be likewise connected on a GND plane.

13.1.2 POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitance should be placed near the Intel486 Microprocessor. The Intel486 Microprocessor driving its 32-bit parallel address and data busses at high frequencies can cause transient power surges, particularly when driving large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the Intel486 Microprocessor and decoupling capacitors as much as possible. Capacitors specifically for PGA packages are also commercially available.

13.1.3 OTHER CONNECTION RECOMMENDATIONS

N.C. pins should always remain unconnected.

For reliable operation, always connect unused inputs to an appropriate signal level. Active LOW inputs should be connected to V_{CC} through a pullup resistor. Pullups in the range of 20 KΩ are recommended. Active HIGH inputs should be connected to GND.

13.2 Maximum Ratings

Table 13.1 is a stress rating only, and functional operation at the maximums is not guaranteed. Function operating conditions are given in 13.3 D.C. Specifications and 13.4 A.C. Specifications.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the Intel486 Microprocessor contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

Table 13.1. Absolute Maximum Ratings

Case Temperature under Bias . . . −65°C to +110°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin with Respect to Ground . . . . . −0.5 to V_{CC} + 0.5V
Supply Voltage with Respect to V_{SS} . . . . . −0.5V to +6.5V

13.3 D.C. Specifications

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

Table 13.2. Intel486™ DX Microprocessor DC Parametric Values (for PGA Package)

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current (50 MHz) Power Supply Current (33 MHz) Power Supply Current (25 MHz)		1000 900 700	mA	(Note 3)
I_{LI}	Input Leakage Current		±15	μA	(Note 4)
I_{IH}	Input Leakage Current		200	μA	(Note 5)
I_{IL}	Input Leakage Current		−400	μA	(Note 6)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance (25 MHz and 33 MHz) (50 MHz)		 20 13	 pF pF	 F_C = 1 MHz (Note 7) F_C = 1 MHz (Note 7)
C_O	I/O or Output Capacitance (25 MHz and 33 MHz) (50 MHz)		 20 17	 pF pF	 F_C = 1 MHz (Note 7) F_C = 1 MHz (Note 7)
C_{CLK}	CLK Capacitance (25 MHz and 33 MHz (50 MHz)		 20 15	 pF pF	 F_C = 1 MHz (Note 7) F_C = 1 MHz (Note 7)

* PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
1. This parameter is measured at:
   Address, Data, BEn 4.0 mA
   Definition, Control 5.0 mA
2. This parameter is measured at:
   Address, Data, BEn −1.0 mA
   Definition, Control −0.9 mA
3. Typical supply current:
   550 mA @ 25 MHz
   700 mA @ 33 MHz
   800 mA @ 50 MHz
4. This parameter is for inputs without internal pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with internal pulldowns and V_{IH} = 2.4V.
6. This parameter is for inputs with internal pullups and V_{IL} = 0.45V.
7. Not 100% tested.

13.4 A.C. Specifications

The A.C. specifications, given in Table 13.3, consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the rising edge of the CLK signal.

A.C. specifications measurement is defined by Figures 13.1–13.7. All timings are referenced to 1.5V unless otherwise specified. Inputs must be driven to the voltage levels indicated by Figure 13.3 when A.C. specifications are measured. Intel486 Microprocessor output delays are specified with minimum and maximum limits, measured as shown. The minimum Intel486 Microprocessor delay times are hold times provided to external circuitry. Intel486 Microprocessor input setup and hold times are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct Intel486 Microprocessor operation.

Table 13.3. 25 MHz Intel486™ Microprocessor A.C. Characteristics (PGA)

V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	25	MHz		1X CLK to Intel486
t_1	CLK Period	40	125	ns	13.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	14		ns	13.1	at 2V(1)
t_3	CLK Low Time	14		ns	13.1	at 0.8V(1)
t_4	CLK Fall Time		4	ns	13.1	(2V − 0.8V)(1)
t_5	CLK Rise Time		4	ns	13.1	(0.8V − 2V)(1)
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	22	ns	13.5	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		30	ns	13.6	(Note 1)
t_8	PCHK# Valid Delay	3	27	ns	13.4	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	27	ns	13.5	
t_9	BLAST#, PLOCK# Float Delay		30	ns	13.6	(Note 1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	22	ns	13.5	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		30	ns	13.6	(Note 1)
t_{12}	EADS# Setup Time	8		ns	13.2	
t_{13}	EADS# Hold Time	3		ns	13.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	8		ns	13.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	13.2	
t_{16}	RDY#, BRDY# Setup Time	8		ns	13.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	13.3	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	10		ns	13.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	13.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	10		ns	13.2	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	13.2	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	5		ns	13.2, 13.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	13.2, 13.3	

PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

NOTE:
1. Not 100% tested. Guaranteed by design characterization.

Table 13.4. 33 MHz Intel486™ Microprocessor A.C. Characteristics (PGA)

V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	33	MHz		1X CLK to Intel486
t_1	CLK Period	30	125	ns	13.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	11		ns	13.1	at 2V(1)
t_3	CLK Low Time	11		ns	13.1	at 0.8V(1)
t_4	CLK Fall Time		3	ns	13.1	(2V − 0.8V)(1)
t_5	CLK Rise Time		3	ns	13.1	(0.8V − 2V)(1)
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	16	ns	13.5	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		20	ns	13.6	(Note 1)
t_8	PCHK# Valid Delay	3	22	ns	13.4	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	20	ns	13.5	
t_9	BLAST#, PLOCK# Float Delay		20	ns	13.6	(Note 1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	18	ns	13.5	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		20	ns	13.6	(Note 1)
t_{12}	EADS# Setup Time	5		ns	13.2	
t_{13}	EADS# Hold Time	3		ns	13.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	5		ns	13.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	13.2	
t_{16}	RDY#, BRDY# Setup Time	5		ns	13.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	13.3	
t_{18}	HOLD, AHOLD, Setup Time	6		ns	13.2	
t_{18a}	BOFF# Setup Time	8		ns	13.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	13.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	5		ns	13.2	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	13.2	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	5		ns	13.2, 13.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	13.2, 13.3	

SEE INTEL PRELIMINARY FOR DESIGN-IN INFORMATION

NOTE:
1. Not 100% tested. Guaranteed by design characterization.

Table 13.5. 50 MHz Intel486™ Microprocessor A.C. Specifications

V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = See Note 2

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	16	50	MHz		1X CLK to Intel486
t_1	CLK Period	20	62.5	ns	13.1	
t_{1a}	CLK Period Stability		0.1%			Adjacent Clocks
t_2	CLK High Time	7		ns	13.1	at 2V(1)
t_3	CLK Low Time	7		ns	13.1	at 0.8V(1)
t_4	CLK Fall Time		2	ns	13.1	(2.0V–0.8V)(1)
t_5	CLK Rise Time		2	ns	13.1	(0.8V–2.0V)(1)
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	12	ns	13.5	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ Float Delay		18	ns	13.6	(Note 1)
t_8	PCHK# Valid Delay	3	14	ns	13.4	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	12	ns	13.5	
t_9	BLAST#, PLOCK# Float Delay		18	ns	13.6	(Note 1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	12	ns	13.5	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		18	ns	13.6	(Note 1)
t_{12}	EADS# Setup Time	5		ns	13.2	
t_{13}	EADS# Hold Time	2		ns	13.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	5		ns	13.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	2		ns	13.2	
t_{16}	RDY#, BRDY# Setup Time	5		ns	13.3	
t_{17}	RDY#, BRDY# Hold Time	2		ns	13.3	
t_{18}	HOLD, AHOLD Setup Time	5		ns	13.2	
t_{18a}	BOFF# Setup Time	5		ns	13.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	2		ns	13.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	5		ns	13.2	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	2		ns	13.2	
t_{22}	D0–D31, DP0–3, A4–A31 Read Data Setup Time	4		ns	13.2, 13.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Data Hold Time	2		ns	13.2, 13.3	

* PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
1. Not 100% tested. Guaranteed by design characterization.
2. Specifications assume C_L = 0 pF. I/O Buffer model must be used to determine delays due to loading (trace and component). First Order I/O buffer models for the Intel486 CPU are available. Contact Intel for the latest release.
3. All timings are referenced at 1.5V (as illustrated in the listed figures) unless otherwise noted.

Table 13.6. 50 MHz Intel486™ Microprocessor A.C. Characteristics for Boundary Scan Test Signals

V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = 50 pF. All Inputs and Outputs are TTL Level

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{24}	TCK Frequency		25	MHz		1x Clock
t_{25}	TCK Period	40		ns		(Note 2)
t_{26}	TCK High Time	10		ns		@ 2.0V
t_{27}	TCK Low Time	10		ns		@ 0.8V
t_{28}	TCK Rise Time		4	ns		(Note 1)
t_{29}	TCK Fall Time		4	ns		(Note 1)
t_{30}	TDI, TMS Setup Time	8		ns	13.7	(Note 3)
t_{31}	TDI, TMS Hold Time	7		ns	13.7	(Note 3)
t_{32}	TDO Valid Delay	3	25	ns	13.7	(Note 3)
t_{33}	TDO Float Delay		TBD	ns		
t_{34}	All Outputs (Non-Test) Valid Delay	3	25	ns	13.7	(Note 3)
t_{35}	All Outputs (Non-Test) Float Delay		36	ns	13.7	(Notes 3, 5)
t_{36}	All Inputs (Non-Test) Setup Time	8		ns	13.7	(Note 3)
t_{37}	All Inputs (Non-Test) Hold Time	7		ns	13.7	(Note 3)

NOTES:
1. Rise/Fall times are measured between 0.8V and 2.0V. Rise/Fall times can be relaxed by 1 ns per 10 ns increase in TCK period.
2. TCK period ≥ CLK period.
3. Parameter measured from TCK.
4. Boundary Scan A.C. Specifications in the above table are target values. They have not been characterized. Therefore they are subject to change.
5. Not 100% tested. Guaranteed by design characterization.

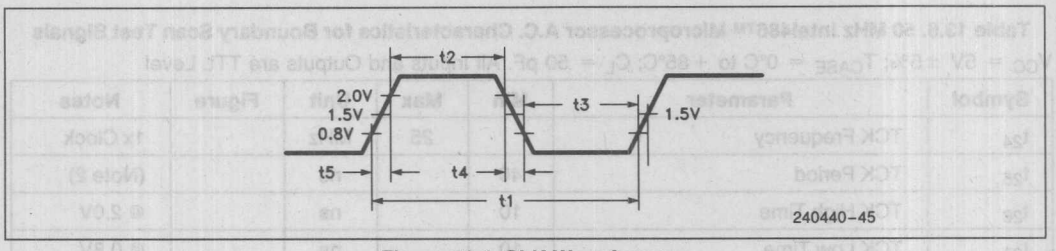


Figure 13.1. CLK Waveforms

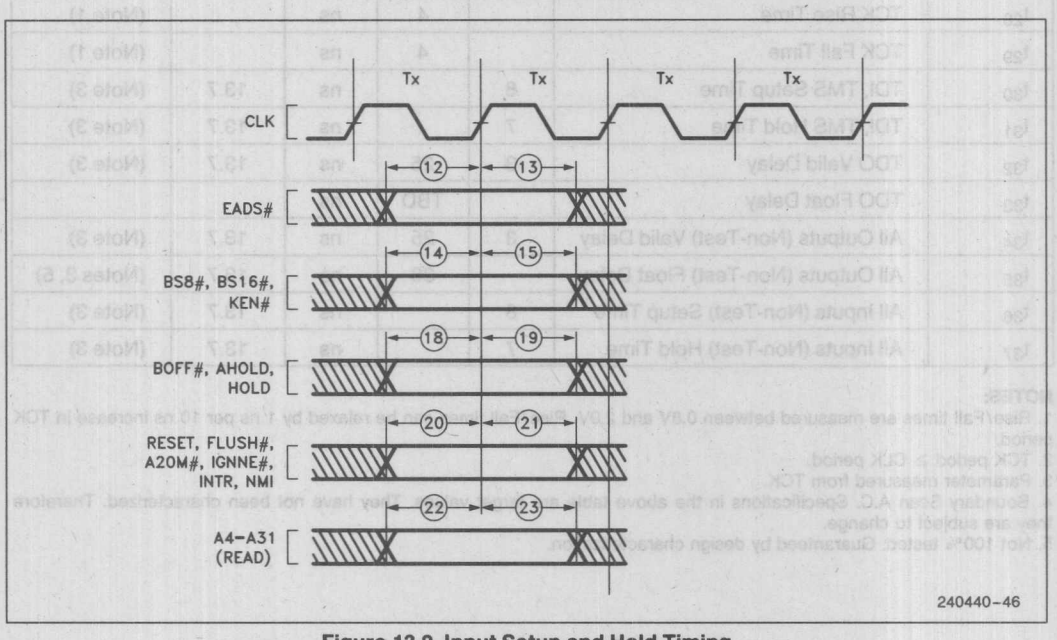


Figure 13.2. Input Setup and Hold Timing

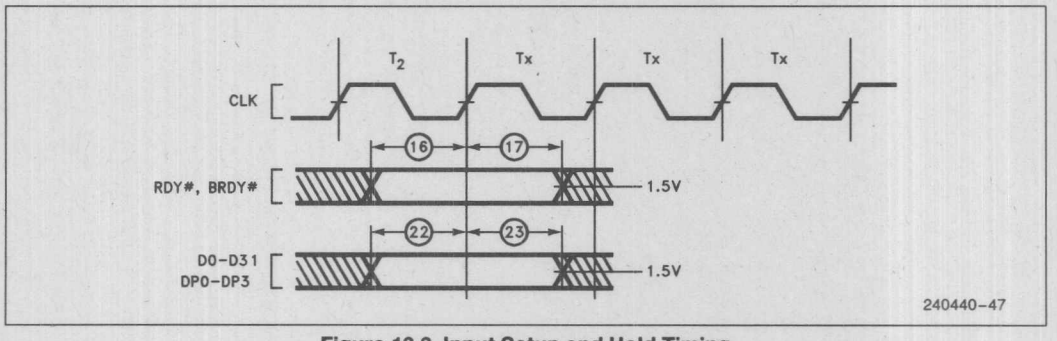


Figure 13.3. Input Setup and Hold Timing

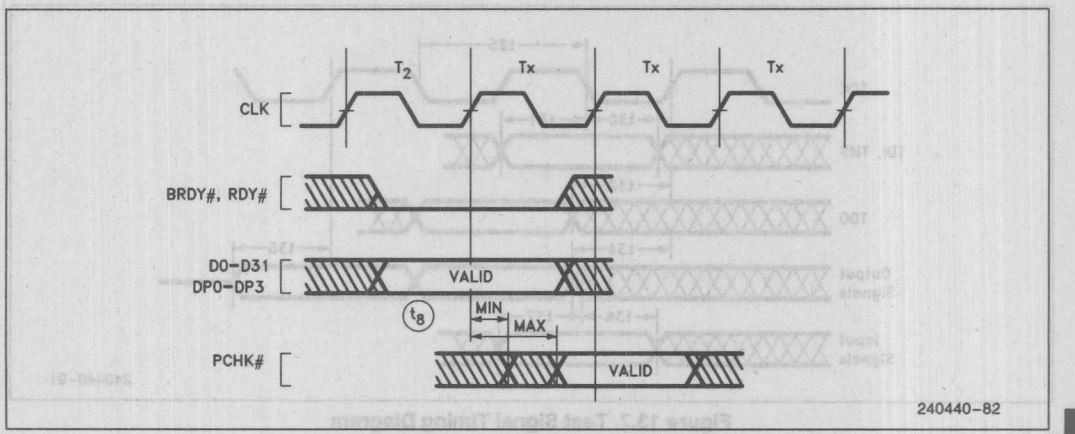


Figure 13.4. PCHK# Valid Delay Timing

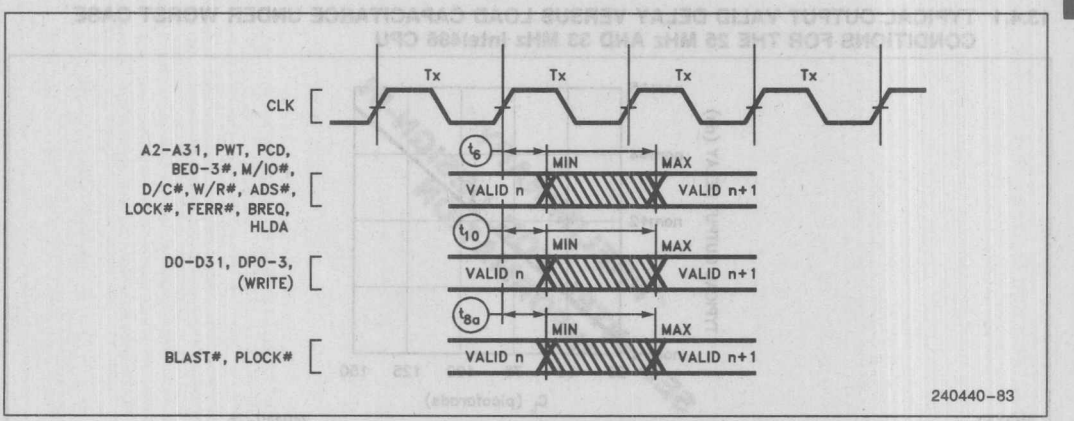


Figure 13.5. Output Valid Delay Timing

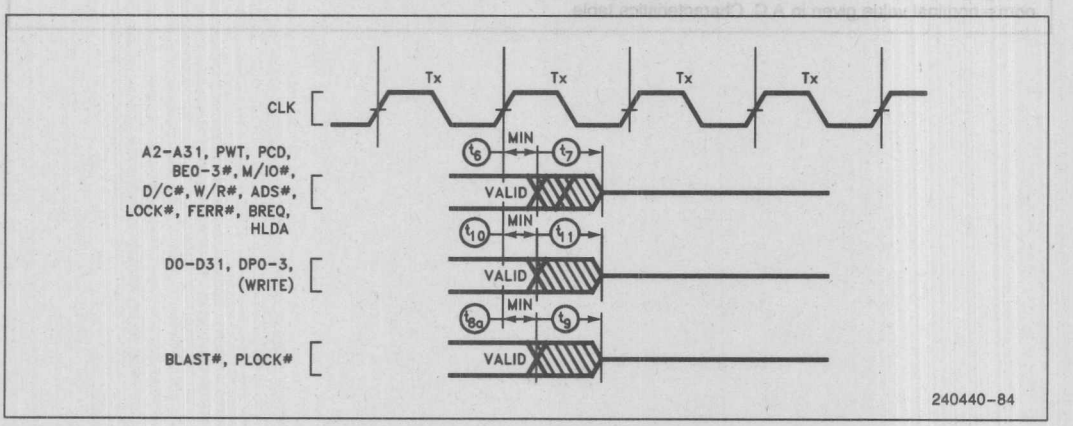


Figure 13.6. Maximum Float Delay Timing

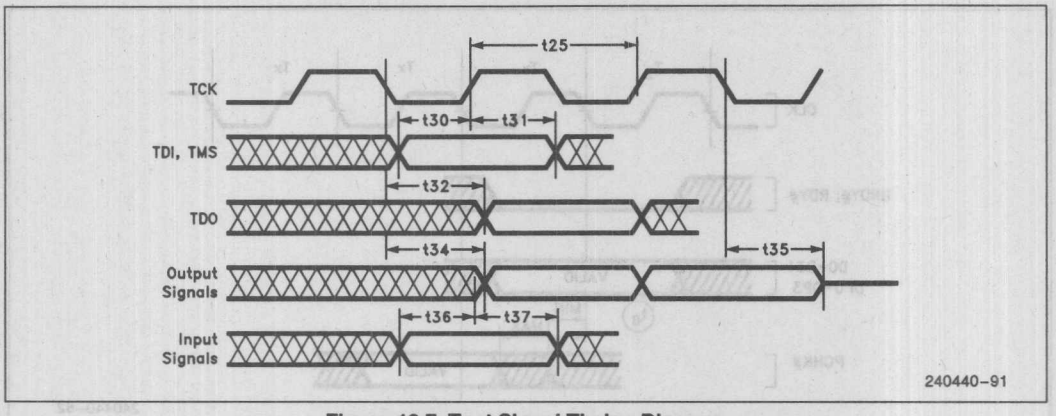
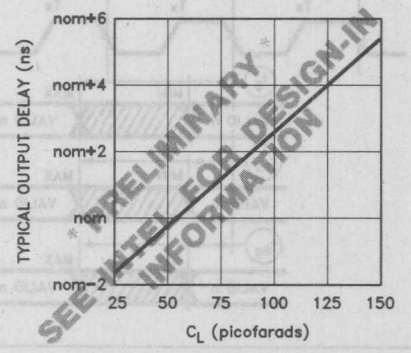


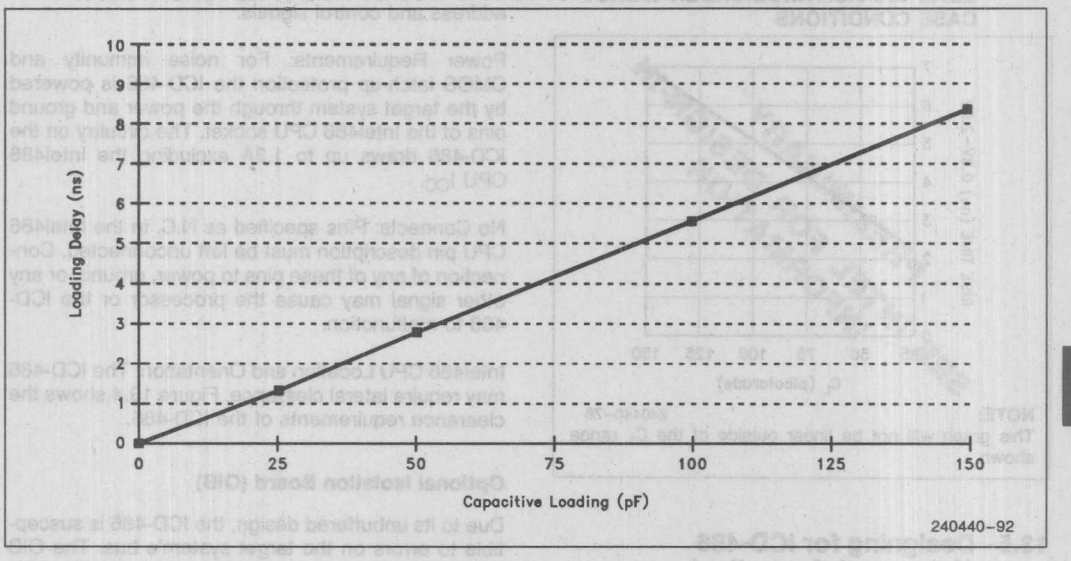
Figure 13.7. Test Signal Timing Diagram

13.4.1 TYPICAL OUTPUT VALID DELAY VERSUS LOAD CAPACITANCE UNDER WORST CASE CONDITIONS FOR THE 25 MHz AND 33 MHz Intel486 CPU

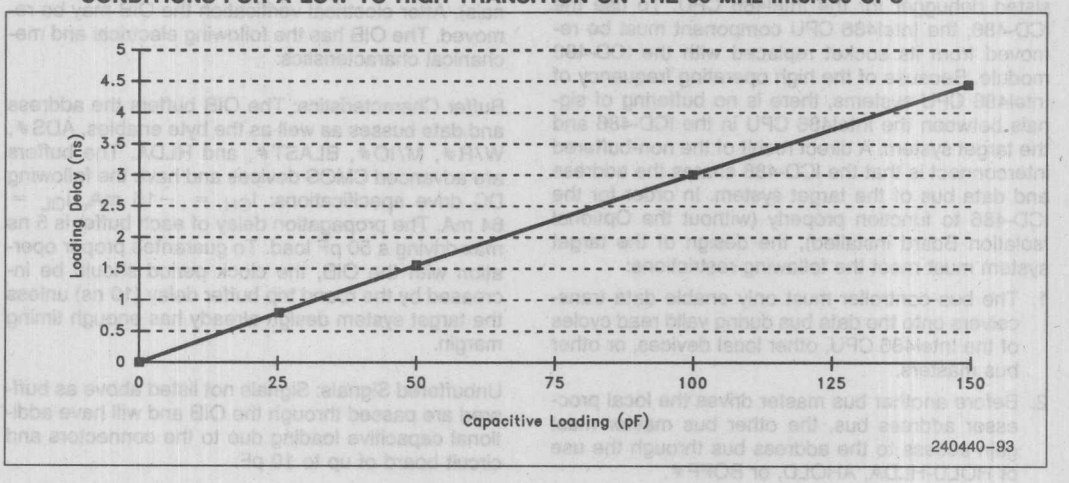


NOTE:
This graph will not be linear outside of the C_L range shown.
nom = nominal value given in A.C. Characteristics table.

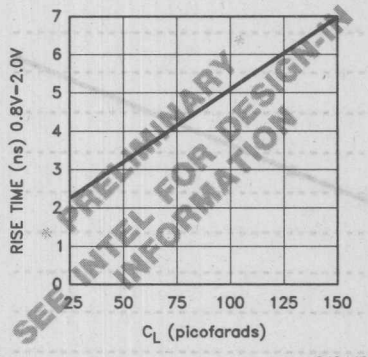
13.4.2 TYPICAL LOADING DELAY VERSUS CAPACITIVE LOADING UNDER WORST-CASE CONDITIONS FOR A HIGH TO LOW TRANSITION ON THE 50 MHz Intel486 CPU



13.4.3 TYPICAL LOADING DELAY VERSUS CAPACITIVE LOADING UNDER WORST-CASE CONDITIONS FOR A LOW TO HIGH TRANSITION ON THE 50 MHz Intel486 CPU



13.4.4 TYPICAL OUTPUT RISE TIME VERSUS LOAD CAPACITANCE UNDER WORST-CASE CONDITIONS



* PRELIMINARY INTEL FOR DESIGN-IN INFORMATION SEE

NOTE:
This graph will not be linear outside of the C_L range shown.

13.5 Designing for ICD-486 (Advance Information)

The ICD-486 (In-Circuit Debugger) is a hardware assisted debugger for the Intel486 CPU. To use the ICD-486, the Intel486 CPU component must be removed from its socket replaced with the ICD-486 module. Because of the high operating frequency of Intel486 CPU systems, there is no buffering of signals between the Intel486 CPU in the ICD-486 and the target system. A direct result of the non-buffered interconnect is that the ICD-486 shares the address and data bus of the target system. In order for the ICD-486 to function properly (without the Optional Isolation Board installed), the design of the target system must meet the following restrictions:

1. The bus controller must only enable data transceivers onto the data bus during valid read cycles of the Intel486 CPU, other local devices, or other bus masters.
2. Before another bus master drives the local processor address bus, the other bus master must gain access to the address bus through the use of HOLD-HLDA, AHOLD, or BOFF#.

In addition to the above restrictions, the ICD-486 has several electrical and mechanical characteristics that should be taken into consideration when designing the Intel486 CPU system.

Capacitive Loading: ICD-486 adds up to 30 pF to the CLK signal, and up to 20 pF to each of the other Intel486 CPU signals.

DC Loading: ICD-486 adds ±15 μA loading to the CLK and data bus signals and ±5 μA loading to the address and control signals.

Power Requirements: For noise immunity and CMOS latch-up protection the ICD-486 is powered by the target system through the power and ground pins of the Intel486 CPU socket. The circuitry on the ICD-486 draws up to 1.3A excluding the Intel486 CPU I_{CC}.

No Connects: Pins specified as N.C. in the Intel486 CPU pin description must be left unconnected. Connection of any of these pins to power, ground, or any other signal may cause the processor or the ICD-486 to malfunction.

Intel486 CPU Location and Orientation: The ICD-486 may require lateral clearance. Figure 13.4 shows the clearance requirements of the ICD-486.

Optional Isolation Board (OIB)

Due to its unbuffered design, the ICD-486 is susceptible to errors on the target system's bus. The OIB installs between the ICD-486 and Intel486 CPU socket in the target system and allows the ICD-486 to function in systems with faults (i.e., shorted signals). After electrical verification the OIB may be removed. The OIB has the following electrical and mechanical characteristics:

Buffer Characteristics: The OIB buffers the address and data busses as well as the byte enables, ADS#, W/R#, M/IO#, BLAST#, and HLDA. The buffers are advanced CMOS devices and have the following DC drive specifications: $I_{OH} = -15$ mA, $I_{OL} = 64$ mA. The propagation delay of each buffer is 5 ns max driving a 50 pF load. To guarantee proper operation with the OIB, the clock period should be increased by the round trip buffer delay (10 ns) unless the target system design already has enough timing margin.

Unbuffered Signals: Signals not listed above as buffered are passed through the OIB and will have additional capacitive loading due to the connectors and circuit board of up to 10 pF.

Power Requirements: The OIB is also powered by the target system through the Intel486 CPU socket and requires 0.5A in addition to the ICD-486 and Intel486 CPU requirements.

OIB Clearance Requirements: The OIB requires an extra 0.55″ of vertical clearance in the target system above the Intel486 CPU socket.

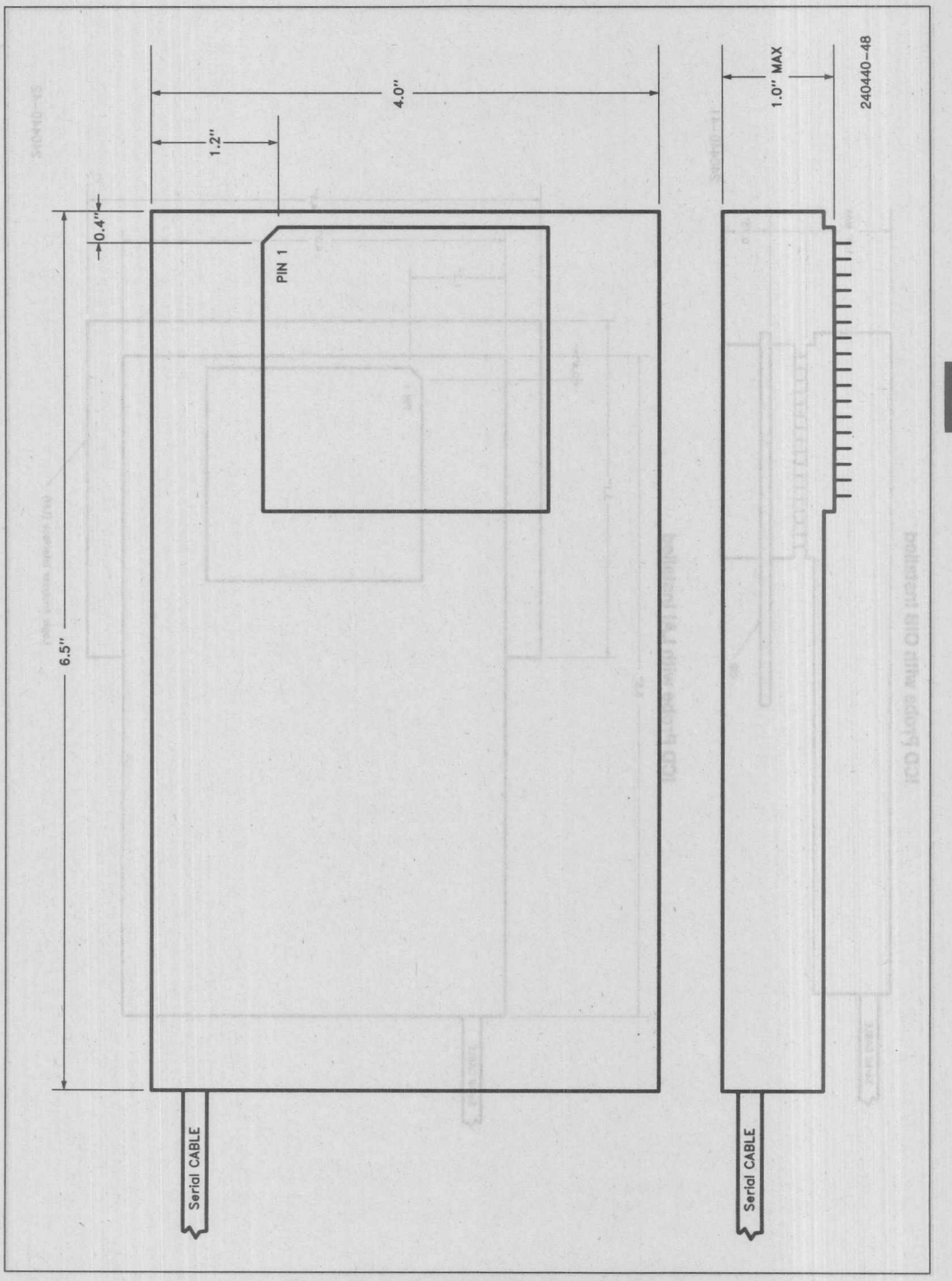


Figure 13.4a. ICD-486 Probe Dimensions

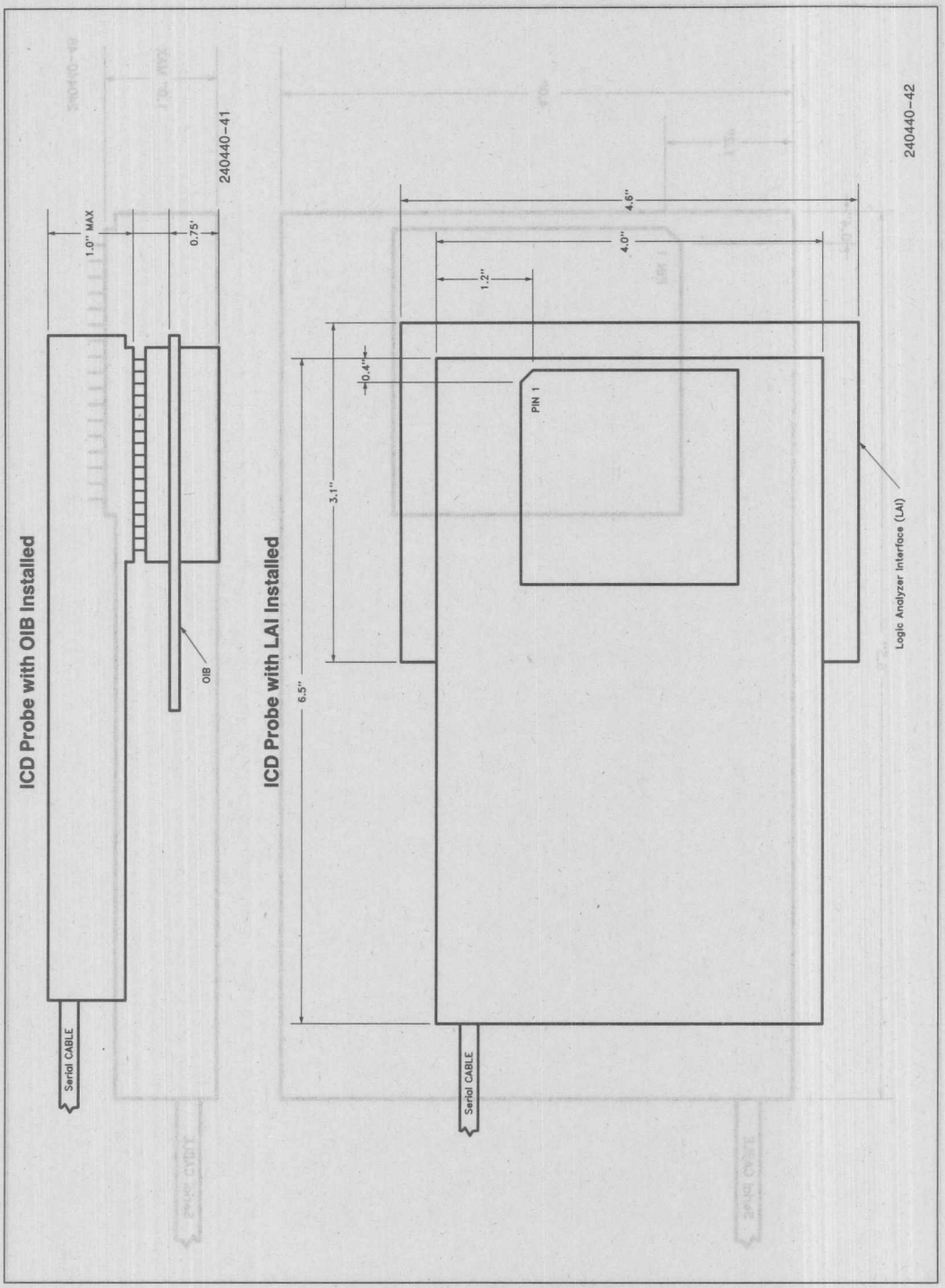
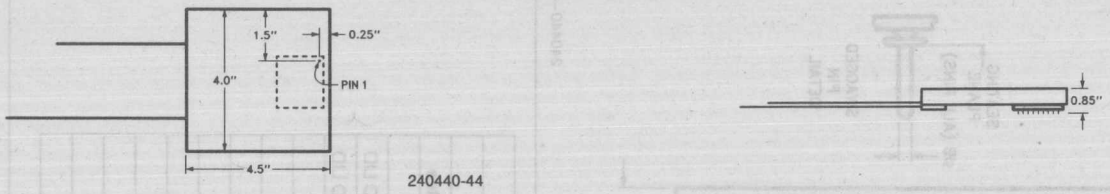


Figure 13.4b. ICD-486 Probe Dimensions

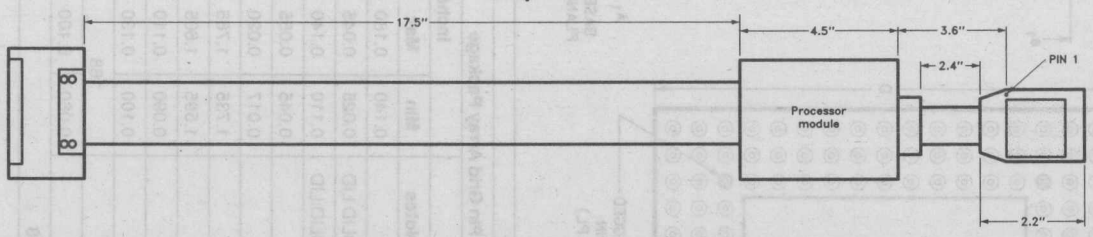
Processor Module Board Dimensions



240440-44

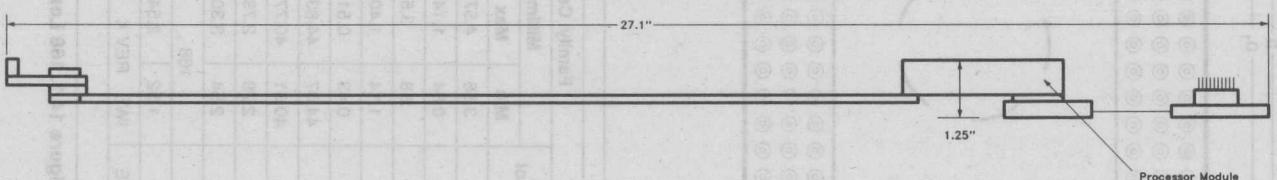
240440-77

Processor Module Assembly Dimensions
Top View



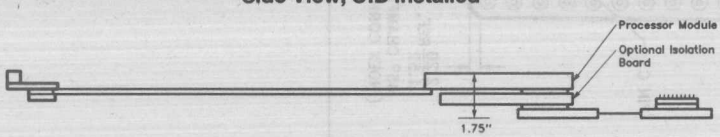
240440-78

Processor Module Assembly Dimensions
Side View



240440-79

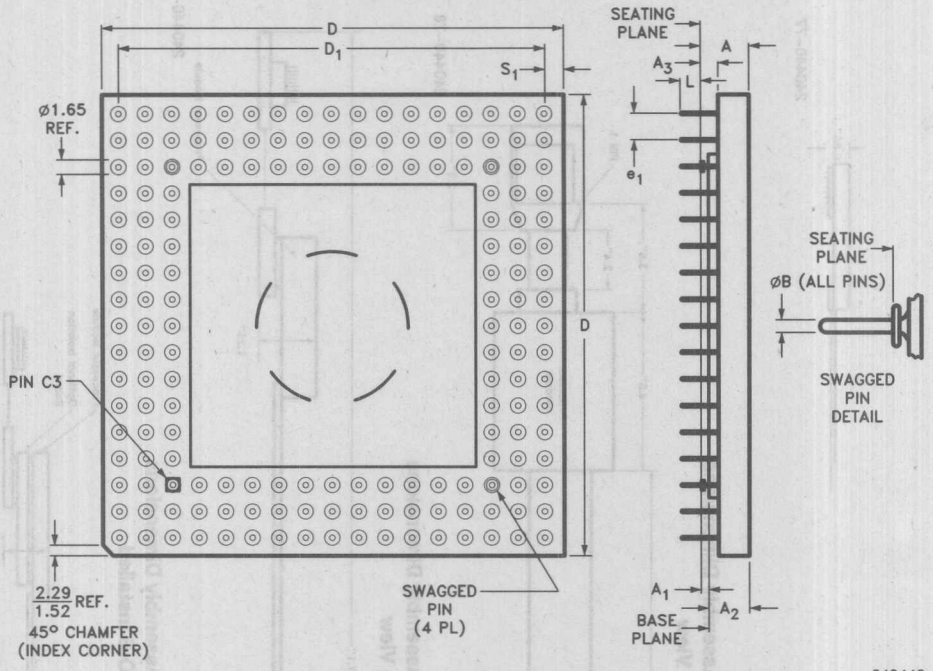
Processor Module Assembly Dimensions
Side View, OIB Installed



240440-80

Figure 13.4c. ICD-486 Probe Dimensions

14.0 MECHANICAL DATA



Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		0.140	0.180	
A_1	0.64	1.14	SOLID LID	0.025	0.045	SOLID LID
A_2	2.8	3.5	SOLID LID	0.110	0.140	SOLID LID
A_3	1.14	1.40		0.045	0.055	
B	0.43	0.51		0.017	0.020	
D	44.07	44.83		1.735	1.765	
D_1	40.51	40.77		1.595	1.605	
e_1	2.29	2.79		0.090	0.110	
L	2.54	3.30		0.100	0.130	
N	168			168		
S_1	1.52	2.54		0.060	0.100	
ISSUE	IWS REV X 7/15/88					

Figure 14.1. 168 Lead Ceramic PGA Package Dimensions

Table 14.1 Ceramic PGA Package Dimension Symbols

Letter or Symbol	Description of Dimensions
A	Distance from seating plane to highest point of body
A_1	Distance between seating plane and base plane (lid)
A_2	Distance from base plane to highest point of body
A_3	Distance from seating plane to bottom of body
B	Diameter of terminal lead pin
D	Largest overall package dimension of length
D_1	A body length dimension, outer lead center to outer lead center
e_1	Linear spacing between true lead position centerlines
L	Distance from seating plane to end of lead
S_1	Other body dimension, outer lead center to edge of body

NOTES:
1. Controlling dimension: millimeter.
2. Dimension "e_1" ("e") is non-cumulative.
3. Seating plane (standoff) is defined by P.C. board hole size: 0.0415–0.0430 inch.
4. Dimensions "B", "B_1" and "C" are nominal.
5. Details of Pin 1 identifier are optional.

14.1 Package Thermal Specifications

The Intel486 Microprocessor is specified for operation when T_C (the case temperature) is within the range of 0°C–85°C. T_C may be measured in any environment to determine whether the Intel486 microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

The ambient temperature (T_A) is guaranteed as long as T_C is not violated. The ambient temperature can be calculated from θ_{JC} and θ_{JA} from the following equations.

$$T_J = T_C + P * \theta_{JC}$$
$$T_A = T_J - P * \theta_{JA}$$
$$T_C = T_A + P * [\theta_{JA} - \theta_{JC}]$$

where T_J, T_A, T_C = Junction, Ambient and Case Temperature respectively. θ_{JC}, θ_{JA} = Junction-to-Case and Junction-to-Ambient Thermal Resistance, respectively.

P = Maximum Power Consumption

The values for θ_{JA} and θ_{JC} are given in Table 14.2 for the 1.75 sq. in., 168-pin, ceramic PGA.

Table 14.3 shows the T_A allowable (without exceeding T_C) at various airflows and operating frequencies (f_{CLK}).

Note that T_A is greatly improved by attaching "fins" or a "heat sink" to the package. P (the maximum power consumption) is calculated by using the maximum I_{CC} at 5V as tabulated in the *DC Characteristics* of Section 13.

Table 14.2.a. Thermal Resistance (°C/W) θ_{JC} and θ_{JA} for the 25 MHz and 33 MHz Intel486 CPU

	θ_{JC}	θ_{JA} vs Airflow—ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
Without Heat Sink	1.5	17	14.5	12.5	11.0	10.0	9.5
With Heat Sink*	2.0	13	8.0	6.0	5.0	4.5	4.25

*0.350" high unidirectional heat sink (Al alloy 6063, 40 mil fin width, 155 mil center-to-center fin spacing).

Table 14.2.b. Thermal Resistance (°C/W) θ_{JC} and θ_{JA} for the 50 MHz Intel486 CPU

	θ_{JC}	θ_{JA} vs Airflow—ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
Without Heat Sink	1.5	16.5	14.0	12.0	10.5	9.5	9.0
With Heat Sink*	2.0	12.0	7.0	5.0	4.0	3.5	3.25

*0.350" high unidirectional heat sink (Al 6063-T5, 40 mil fin width, 155 mil center to center fin spacing).

Heat Sink Dimensions

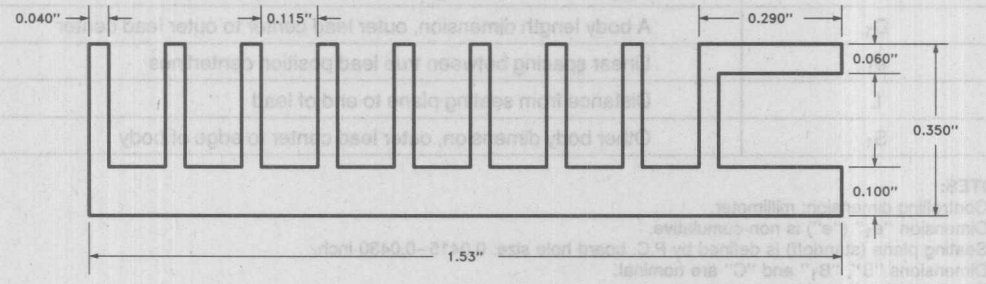


Table 14.3. Maximum T_A at Various Airflows In °C

		Airflow-ft/min (m/sec)					
	f_{CLK} (MHz)	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
T_A with Heat Sink	25.0	47	64	71	75	76	77
	33.3	36	58	67	72	74	75
	50	35	60	70	75	77.5	78.75
T_A without Heat Sink	25.0	31	40	47	52	55	57
	33.3	15	27	36	42	47	49
	50	10	22.5	32.5	40	45	47.5

15.0 LOW POWER Intel486™ DX MICROPROCESSOR

- Lower Power Dissipation
  - — Dynamic Frequency Scalability
  - — I_{CC}(max) Reduced to 150 mA at 2 MHz
  - — Improved V_{CC} Rating (±10%)
- 168-Lead Pin Grid Array Package for Intel486 DX Microprocessor
- High Performance Design
  - — 25 MHz Operation for Intel486™ DX
  - — 64 MByte/Sec Burst Bus
  - — CHMOS IV Process Technology
  - — Dynamic Bus Sizing for 8-, 16- and 32-Bit Buses

This section describes the Low Power Intel486 DX microprocessor.

The Low Power Intel486 Microprocessor meets today's need for high performance portables. The combination of special features like dynamic frequency scaling, lower minimum frequency, improved V_{CC} operation and high integration contribute significantly to lower power dissipation and meet the needs of portable computing.

The Low Power capability is achieved by operating the Intel486 Microprocessor in the 2X mode. The frequency can be varied dynamically between maximum to minimum as needed. The frequency change does not affect contents of the registers and data integrity is maintained. Power dissipation is reduced significantly at 2 MHz where I_{CC} is only 150 mA compared to 600 mA at 20 MHz. Low power versions are offered for both the Intel486 SX and the Intel486 DX Microprocessors.

The Low Power Intel486 Microprocessor is 100-percent compatible with all versions of the Intel386™ Microprocessor family, assuring compatibility with the more than $50 billion software base of MS-DOS, Windows, OS/2 and UNIX/System operating system applications. The Low Power Intel486 Microprocessor integrates the same RISC-technology, one clock per instruction integer core, on-chip cache, and memory management unit as the standard Intel486 Microprocessor.

Note that the Intel OverDrive™ Processor does not work in systems based on the Low Power Intel486 CPU.

The following section on the Low Power Intel 486 DX Microprocessor contains information specific to the Low Power device only. All data not defined are located in the appropriate sections of this data sheet unless specified otherwise.

15.1 Introduction

The Low Power Intel486 Microprocessor brings Intel486 technology and performance to the portable computer market. The low power capability is achieved by a frequency scalability feature during normal operation. The operating frequency can be brought down dynamically resulting in lower power supply current (I_{CC}). This results in minimal power dissipation which ensures a longer battery life.

The Low Power Intel486 Microprocessor integrates the same RISC-technology, one clock per instruction integer core, on-chip cache, and memory management unit as the standard Intel486 Microprocessor.

The Low Power Intel486 Microprocessor has the following special features:

- **Frequency Scalability**—This is achieved by operating the Intel486 Microprocessor in the 2X clock mode. The frequency can be varied dynamically from maximum back to minimum or vice versa. The frequency change does not affect the register content of the CPU, thus data integrity is maintained.
- **Lower Minimum Frequency**—The Low Power Intel486 Microprocessor can be operated at a minimum frequency of 2 MHz, at which I_{CC}(max) is only 150 mA, compared to an I_{CC}(max) of 600 mA at 20 MHz operation. The power dissipation is thus drastically reduced ensuring a longer battery life.
- **Improved V_{CC} Operation**—The Low Power Intel486 Microprocessor has an improved V_{CC} rating of ±10%. Again this feature makes it extremely attractive to portable battery powered applications.

The above three features ensure power savings for portable computer systems resulting in prolonged battery life.

Besides the above special features, the Low Power Intel486 Microprocessor has an identical feature set to the standard Intel486 CPU. This includes:

- **Binary Compatibility**—The Low Power Intel486 CPU is binary compatible with the 8086, 8088, 80186, 80286, Intel386 SX, Intel386 DX, Intel486 SX and Intel486 DX CPUs.

- **Full 32-Bit Integer Processor**—The Low Power Intel486 CPU performs a complete set of arithmetic and logical operations on 8-, 16-, and 32-bit data types using a full-width ALU and eight general-purpose registers.
- **Separate 32-Bit Address and Data Paths**—Four gigabytes of physical memory can be addressed directly.
- **Single-Cycle Execution**—Many instructions execute in a single clock cycle.
- **On-Chip Floating Point Unit**—The 32-, 64-, and 80-bit formats specified in IEEE standard 754 are supported. The unit is binary compatible with the 8087, 80287, Intel387™, Intel387 SX, and Intel487™ Math Coprocessors and the Intel486™ CPU.
- **On-Chip Memory Management Unit**—Address-management and memory-space protection mechanisms maintain the integrity of memory. This is necessary in multitasking and virtual-memory environments, like those implemented by the UNIX and OS/2 operating systems. Both memory segmentation and paging are supported.
- **On-Chip Cache with Cache Consistency Support**—The internal write-through cache can hold 8 KBytes of data or instructions. Cache hits are as fast as read accesses to a processor register. Bus activity is tracked to detect alterations in the memory which internal cache represents. The internal cache can be invalidated or flushed, so that an external cache controller can maintain cache consistency in multi-processor environments.
- **External Cache Control**—Write-back and flush controls over an external cache are provided so that the processor can maintain cache consistency in multi-processor environments.
- **Instruction Pipelining**—The fetching, decoding, execution and address translation of instructions are overlaped within the Low Power Intel486 Microprocessor. This results in a continuous execution rate of one clock cycle per instruction, for most instructions.
- **Burst Cycles**—Burst transfers allow a new doubleword to be read from memory each clock cycle. With this capability the internal cache and instruction prefetch buffer can be filled very rapidly.
- **Write Buffers**—The processor contains write buffers to enhance the performance of consecutive writes to memory. The Low Power Intel486 CPU can continue operations internally after a write, without waiting for the write to be executed on the external bus.
- **Bus Backoff**—If another bus master needs control of the bus during a Low Power Intel486 Microprocessor initiated bus cycle, the Low Power Intel486 Microprocessor will float its bus signals, then restart its cycle when the bus becomes available again.
- **Instruction Restart**—Programs can continue execution following an exception generated by an unsuccessful attempt to access memory. This feature is important for supporting demand-paged virtual memory applications.
- **Dynamic Bus Sizing**—External controllers can dynamically alter the effective width of the data bus. Bus widths of 8, 16 or 32 bits can be used.

The Low Power Intel386 DX Microprocessor pinout follows the same definition as the Intel486 DX Microprocessor given in Section 1.0 except for those listed in Table 15.1.

Table 15.1

i486 DX Microprocessor	Low Power i486 DX Microprocessor	Pin #
CLK	CLK2	C3
NC	CLKSEL	A3(1)

NOTE:
1. This pin is TCK on the 50 MHz Intel486 DX Microprocessor.

15.2 Pinout

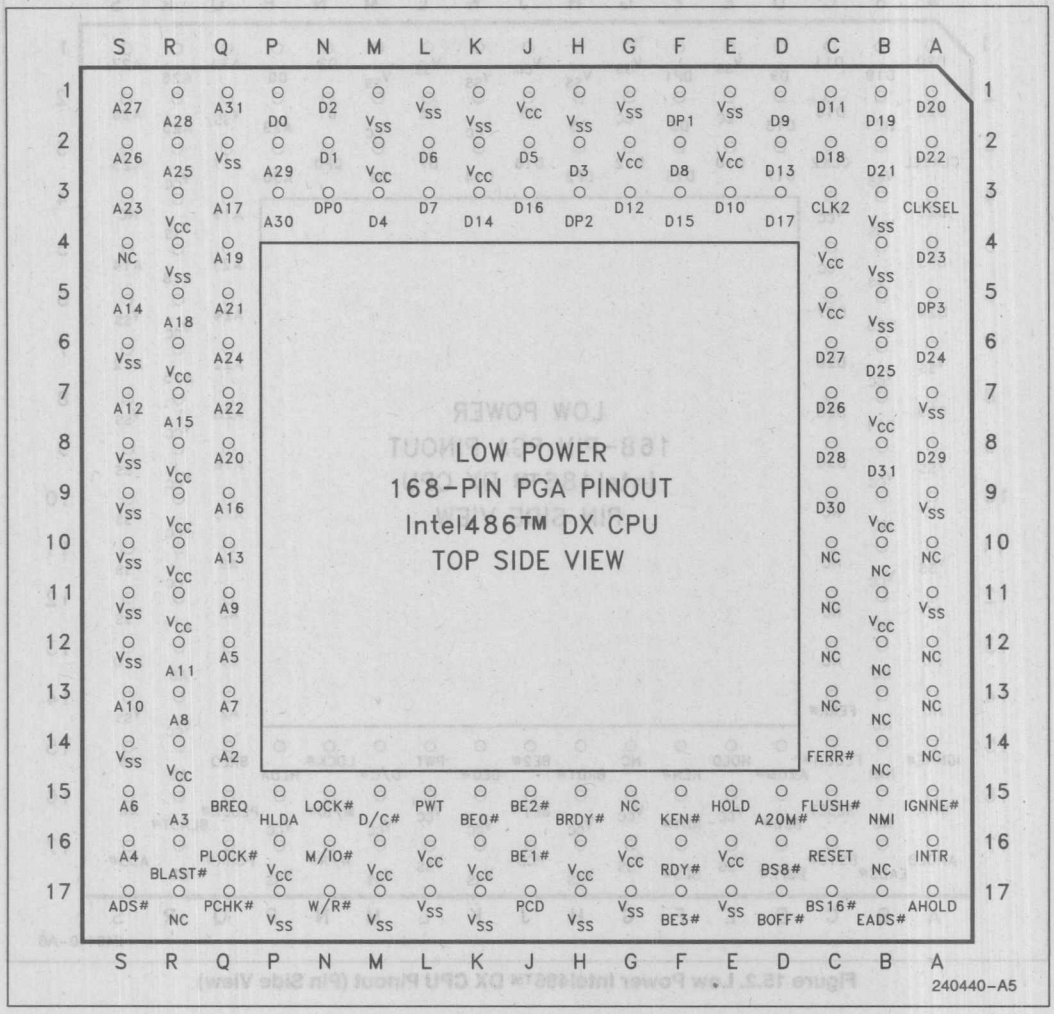


Figure 15.1. Low Power Intel486™ DX CPU Pinout (Top Side View)

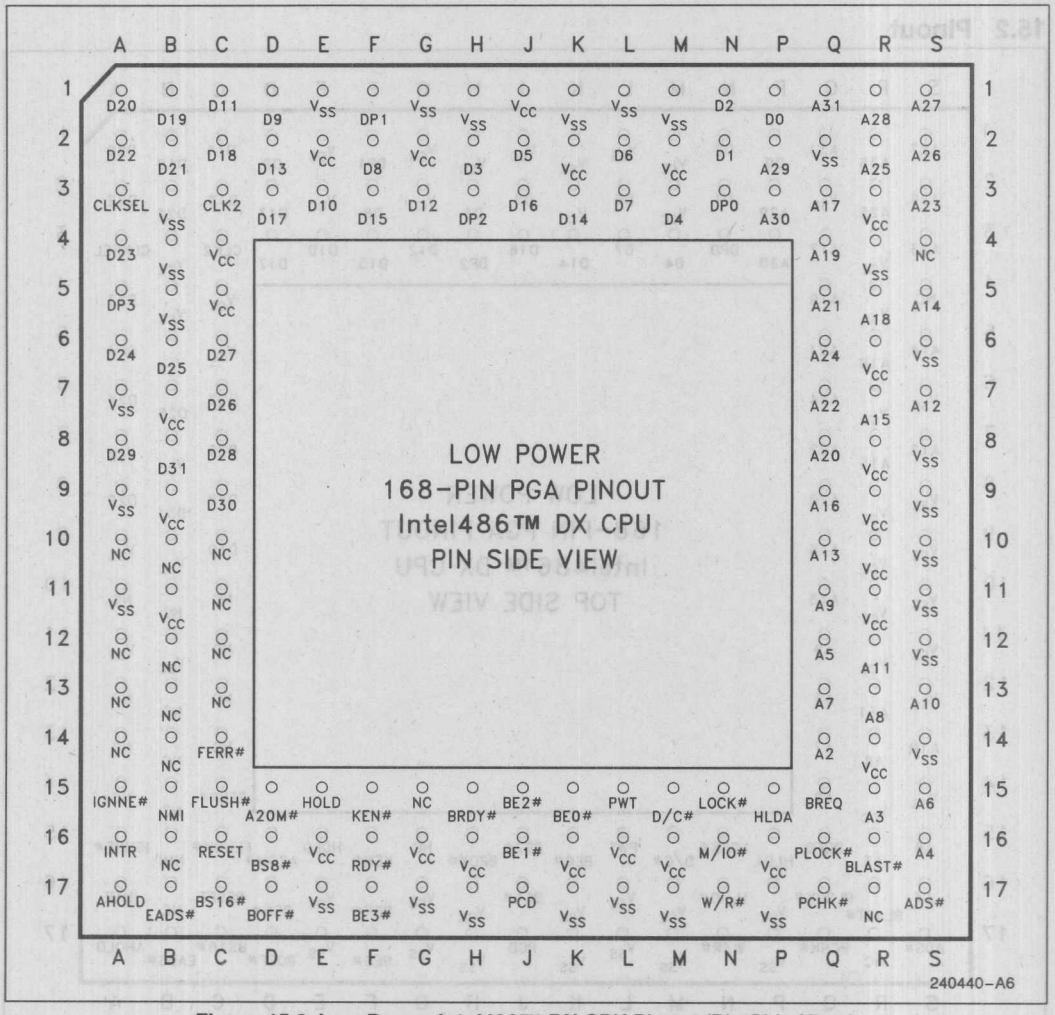


Figure 15.2. Low Power Intel486™ DX CPU Pinout (Pin Side View)

15.3 Pin Cross Reference (Intel486™ DX CPU)

Address		Data		Control		N/C	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	A10	B7	A7
A_3	R15	D_1	N2	ADS#	S17	A12	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	A13	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	A14	C4	B3
A_6	S15	D_4	M3	BE1#	J16	B10	C5	B4
A_7	Q13	D_5	J2	BE2#	J15	B12	E2	B5
A_8	R13	D_6	L2	BE3#	F17	B13	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16	B14	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17	B16	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15	C10	H16	G17
A_{12}	S7	D_{10}	E3	BREQ	Q15	C11	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16	C12	K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17	C13	K16	K1
A_{15}	R7	D_{13}	D2	CLK2	C3	G15	L16	K17
A_{16}	Q9	D_{14}	K3	CLKSEL	A3	R17	M2	L1
A_{17}	Q3	D_{15}	F3	D/C#	M15	S4	M16	L17
A_{18}	R5	D_{16}	J3	DP0	N3		P16	M1
A_{19}	Q4	D_{17}	D3	DP1	F1		R3	M17
A_{20}	Q8	D_{18}	C2	DP2	H3		R6	P17
A_{21}	Q5	D_{19}	B1	DP3	A5		R8	Q2
A_{22}	Q7	D_{20}	A1	EADS#	B17		R9	R4
A_{23}	S3	D_{21}	B2	FERR#	C14		R10	S6
A_{24}	Q6	D_{22}	A2	FLUSH#	C15		R11	S8
A_{25}	R2	D_{23}	A4	HLDA	P15		R14	S9
A_{26}	S2	D_{24}	A6	HOLD	E15			S10
A_{27}	S1	D_{25}	B6	IGNNE#	A15			S11
A_{28}	R1	D_{26}	C7	INTR	A16			S12
A_{29}	P2	D_{27}	C6	KEN#	F15			S14
A_{30}	P3	D_{28}	C8	LOCK#	N15			
A_{31}	Q1	D_{29}	A8	M/IO#	N16			
		D_{30}	C9	NMI	B15			
		D_{31}	B8	PCD	J17			
				PCHK#	Q17			
				PWT	L15			
				PLOCK#	Q16			
				RDY#	F16			
				RESET	C16			
				W/R#	N17			

15.4 Pin Description

All pin descriptions for the Low Power Intel486 DX Microprocessor follow the same definition as the Intel486 DX Microprocessor with the exception of those listed in Table 15.2.

Table 15.2

Symbol	Type	Name and Function
CLK2	I	**CLK2** provides the fundamental timing for the Low Power Intel486 DX Microprocessor. This is twice the internal frequency of the CPU.
CLKSEL	I	**Clock Select** pin selects the 2X mode required for the Low Power Intel486 CPU. A well defined pulse on this pin establishes the phase relationship of the 2X clock. With the exception of a pulse during cold reset, this pin should be driven low at all times and must be free of spikes or glitches.

OUTPUT PINS

Table 15.3. lists all the output pins, indicating their active level, and when they are floated.

Table 15.3. Output Pins

Name	Active Level	When Floated
BREQ	HIGH	
HLDA	HIGH	
BE0#–BE3#	LOW	Bus Hold
PWT, PCD	HIGH	Bus Hold
W/R#, D/C#, M/IO#	HIGH/LOW	Bus Hold
LOCK#	LOW	Bus Hold
PLOCK#	LOW	Bus Hold
ADS#	LOW	Bus Hold
BLAST#	LOW	Bus Hold
PCHK#	LOW	
FERR#	LOW	
A2-A3	HIGH	Bus, Address Hold

INPUT PINS

Table 15.4 lists all input pins, indicating their active level, and whether they are synchronous or asynchronous inputs.

Table 15.4. Input Pins

Name	Active Level	Synchronous/ Asynchronous
CLK2		
CLKSEL		
RESET	HIGH	Asynchronous
HOLD	HIGH	Synchronous
AHOLD	HIGH	Synchronous
EADS#	LOW	Synchronous
BOFF#	LOW	Synchronous
FLUSH#	LOW	Asynchronous
A20M#	LOW	Asynchronous
BS16#, BS8#	LOW	Synchronous
KEN#	LOW	Synchronous
RDY#	LOW	Synchronous
BRDY#	LOW	Synchronous
INTR	HIGH	Asynchronous
NMI	HIGH	Asynchronous
IGNNE#	LOW	Asynchronous

INPUT/OUTPUT PINS

Table 15.5 lists all the input/output pins, indicating their active level and when they are floated.

Table 15.5. Input/Output Pins

Name	Active Level	When Floated
D0–D31	HIGH	Bus Hold
DP0–DP3	HIGH	Bus Hold
A4–A31	HIGH	Bus, Address Hold

Table 15.6. Test Pins

Name	Input or Output	Sampled/ Driven On
TCK	Input	N/A
TDI	Input	Rising Edge of TCK
TDO	Output	Falling Edge of TCK
TMS	Input	Rising Edge of TCK

Table 15.7. Component and Revision ID (PGA)

i486 SX Microprocessor Low Power Stepping Name	Component ID	Revision ID
D0	04	04

NOTE:
Table 15.7 shows the Component ID number and Revision ID number for the D-0 stepping of the Intel486 DX Microprocessor. When an Intel OverDrive Processor is installed in the system, the Component ID and Revision ID is provided by the OverDrive Processor and not the Intel486 DX Microprocessor. The Component ID and Revision ID read by the BIOS/software may change when a Performance Upgrade Component, such as the Intel OverDrive Processor, is installed in an Intel486 DX Microprocessor based system.

15.5 Signal Description

With the exception of CLK2 and CLKSEL, all signals follow the same definition as the Intel486 Microprocessor. The A.C. timing parameters for all of these signals are given in Table 15.11.

CLOCK (CLK2)

CLK2 provides the fundamental timing for the Low Power Intel486 Microprocessor. It is divided by two internally to generate the internal processor clock used for instruction execution. The internal clock is comprised of two phases, "phase one" and "phase two". Each CLK2 period is a phase of the internal clock. Figure 15.3 illustrates the relationship. If desired, the phase of the internal processor clock can be synchronized to a known phase by ensuring the pulse on the CLKSEL pin meets the applicable timings during cold boot (power-up reset).

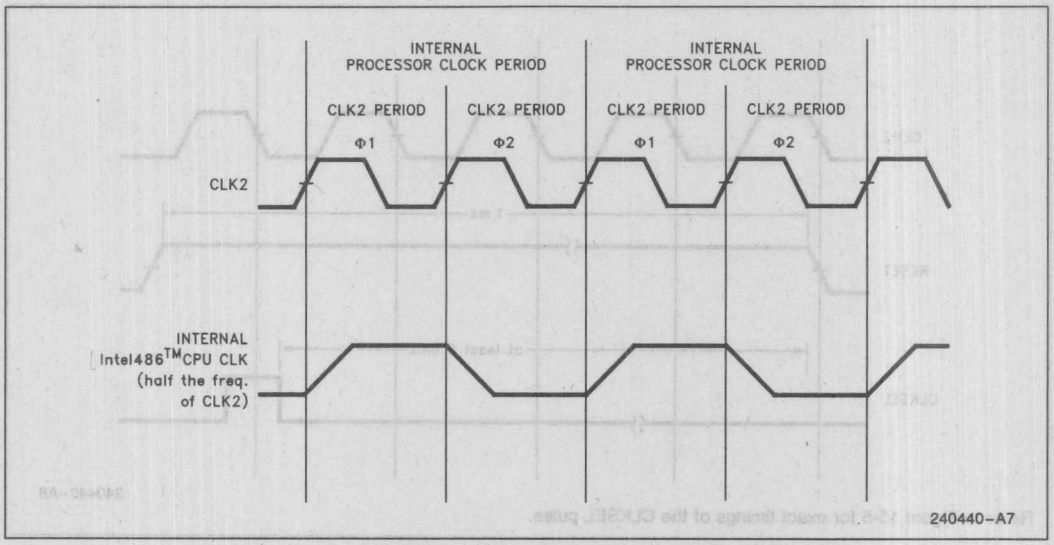


Figure 15.3. CLK2 Signal and Internal Processor Clock

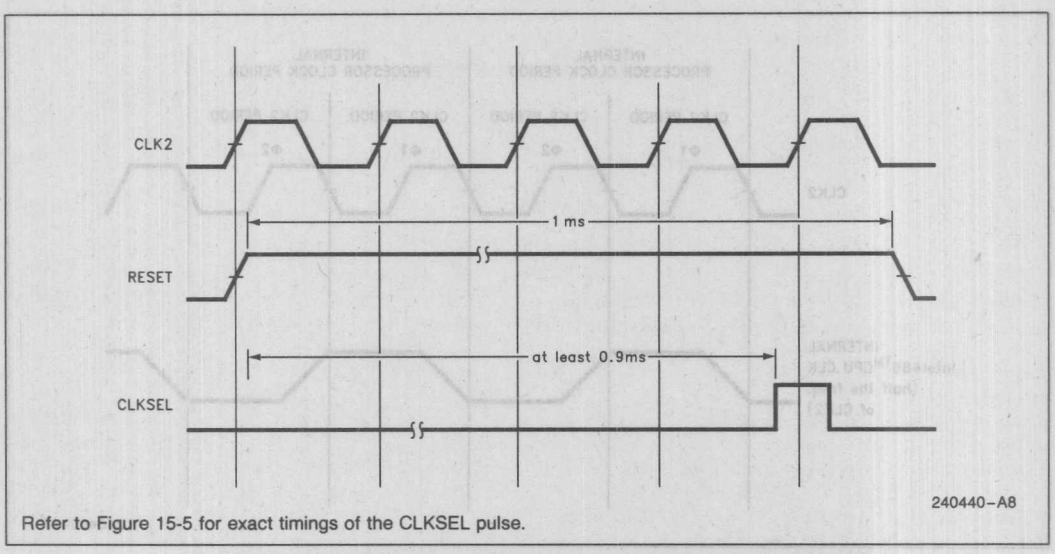
All set-up, hold, float-delay and valid delay timings are referenced to the phase one of the clock.

The internal processor clock (CLK) is similar to the clock signal of the standard Intel486 Microprocessor. All I/O signals get sampled on the rising edge of this signal, i.e. the rising edge of phase one. Thus it is important to synchronize the external circuitry with the phase one of CLK2.

CLKSEL

Clock Select pin selects the 2X mode required for the Low Power Intel486 DX CPU. This pin should be driven low after power-up and during the entire operation of the CPU. However, a well defined pulse is required on CLKSEL pin during cold boot (power-up reset) to establish the phase relationship of the 2X clock. The reset pulse width during cold reset should be at least 1 ms. As shown in Figure 15.4, the pulse on CLKSEL should be asserted by the end of reset (approximately 0.9 ms after driving reset active) and at least 30 CLK2 periods before the falling edge of reset.

Figure 15.5 shows the detailed timing definition of this pulse. The pulse on CLKSEL pin is only required during power-up reset. During all other times including warm resets the CLKSEL pin should be driven low and must be free of spikes or glitches. After the power-up reset, the system must track the phase of CLK2 at all times including during warm resets so that the input/output signals can be sampled at the appropriate clock edge. The phase relationship is described in the next section.



Refer to Figure 15-5 for exact timings of the CLKSEL pulse.

Figure 15.4. CLKSEL Pulse with Reference to the Reset Pulse Width

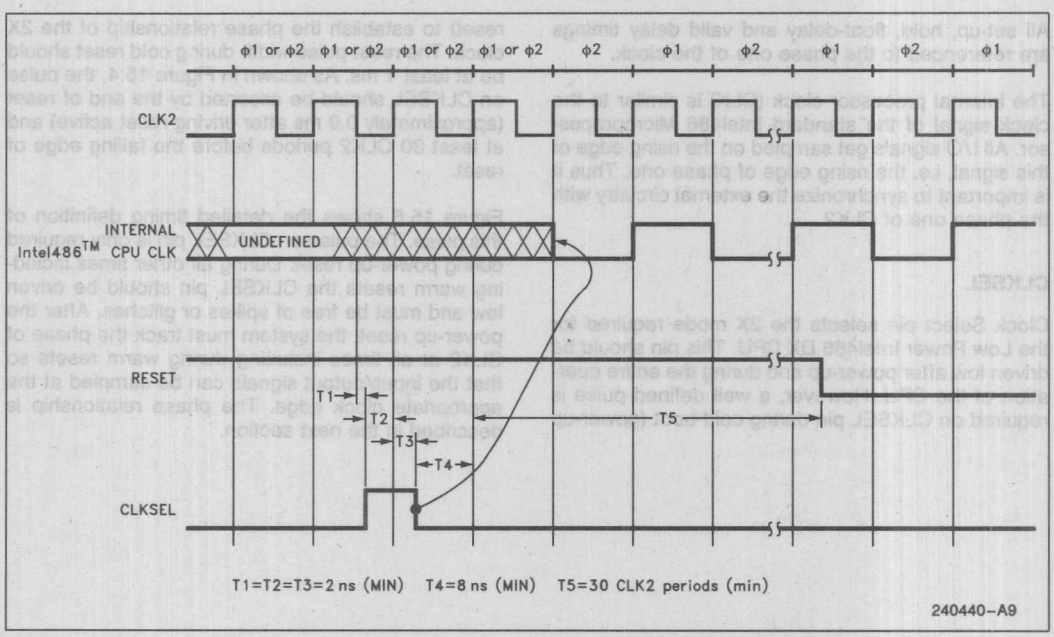


Figure 15.5. CLKSEL Timing Definition during Power-Up Reset

15.6 Architecture Overview

The Low Power Intel486 DX Microprocessor is architecturally similar to the Intel486 CPU. Thus all bus cycles follow the same definition The difference lies in the fact that the Low Power Intel486 CPU works with an external 2X clock input (CLK2). As shown in Figure 16-3, each of the internal processor clock (CLK) cycle is two CLK2 cycles wide. Thus a 25 MHz Low Power Intel486 DX Microprocessor needs a 50 MHz clock input.

CLK2 provides the fundamental timing for the Low Power Intel486 CPU. It is divided by two internally to generate the internal processor clock (CLK) used for instruction execution. The internal clock is comprised of two phases, "phase one" and "phase two". Each CLK2 period is a phase of the internal clock. All Low Power Intel486 Microprocessor inputs are sampled at the rising edge of phase 1. Each bus cycle is comprised of at least two bus states, T1 and T2. Each bus state in turn consists of two CLK2 cycles phase 1 and phase 2 of the bus state. The bus state diagram in Section 7.2.13 is valid for the Low Power Intel486 Microprocessor.

NOTE:
The timing diagrams given in the Intel486 data sheet can be used for the Low Power Intel486 Microprocessor. Read "CLK" signal as the internal clock of the CPU, with "CLK2" (the input clock of the Low Power Intel486 CPU) being twice the frequency of the internal processor clock as shown in Figure 15.3.

The following describes how the input signals are sampled and output signals are referenced with respect to the input clock (CLK2):

INPUT SIGNALS:

The Low Power Intel486 CPU samples all its **synchronous** input signals (i.e. RDY#, BRDY#, BS8#, BS16#, KEN#, EADS#, BOFF#, HOLD and AHOLD) at the rising edge of phase 1, as long as proper setup and hold times relative to that clock edge are met.

The Low Power Intel486 CPU samples all its **asynchronous** input signals (i.e. RESET, INTR, NMI, A20M# FLUSH#, IGNNE#) at every other rising edge of the system clock (Phase 1), as long as proper setup and hold times relative to that clock edge are met.

OUTPUT SIGNALS

The A.C. timing specifications for output signals (i.e. valid and float delay timings) are specified with respect to the rising edge of the Phase 1 of the system clock. This holds true for all output signals including ADS# and PCHK#.

15.7 Variable CPU Frequency

The Low Power Intel486 Microprocessor allows the CPU frequency to change dynamically. As shown in Figures 15.6 and 15.7, the relationship between frequency and power consumption is approximately linear. Thus lowering the CPU frequency, reduces the power supply current (I_{CC}) consumed by the CPU.

The following must be satisified to change the CPU frequency:

1. Frequency can be changed at least 8 clocks after satisfying t4 (see Figure 15.5). The system can be started at a lower frequency and after satisfying the CLKSEL pulse specifications, it can be operated at the required speed.
2. The change in frequency should satisfy the minimum specification of "CLK2 high time" and "CLK2 low time". That is, at no time should the clock period go below the specified clock high and clock low times (see A.C. specifications for exact values).

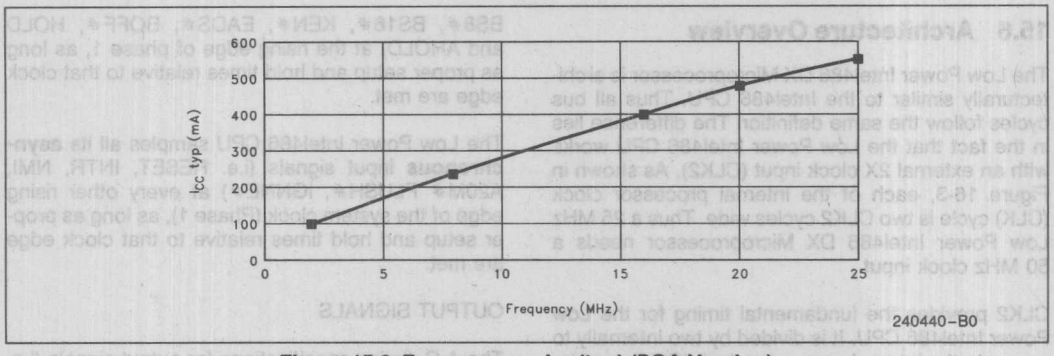


Figure 15.6. Frequency vs I_{CC}(typ) (PGA Version)

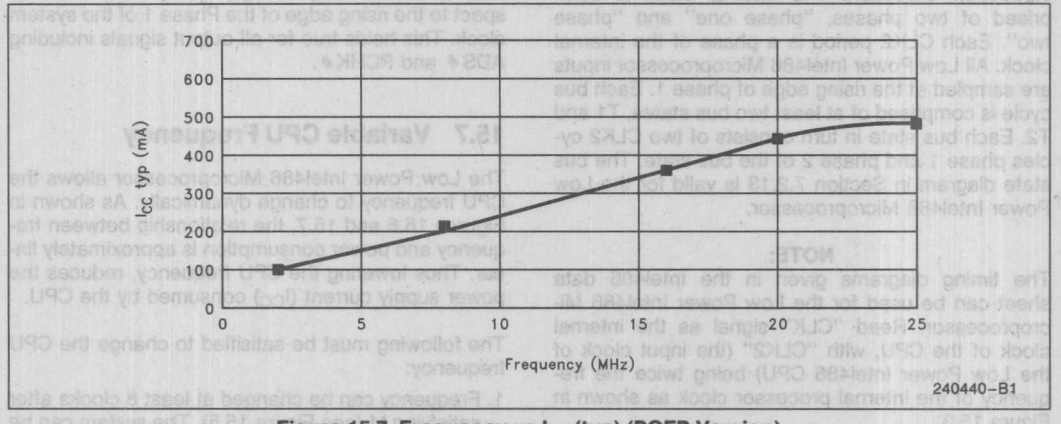


Figure 15.7. Frequency vs I_{CC}(typ) (PQFP Version)

15.8 D.C./A.C. Specifications

Table 15.8 provides the absolute maximum ratings. It is a stress rating only and functional operation at the maximums is not guaranteed. Functional operating conditions are given in Section 15.8.1 D.C. Specifications and Section 15.8.3 A.C. Specifications.

Table 15.8. Absolute Maximum Ratings

Case Temperature under Bias	−65°C to +110°C
Storage Temperature	−65°C to +150°C
Voltage on Any Pin with Respect to Ground	−0.5V to (V_{CC} +0.5V)
Supply Voltage with Respect to V_{SS}	−0.5V to +6.5V

15.8.1 D.C. SPECIFICATIONS

Table 15.9 provides the D.C. operating conditions for the Low Power Intel486 DX Microprocessor.

Functional operating range: V_{CC} = 5V ±10%; T_{CASE} = 0°C to +85°C.

Table 15.9. Low Power Intel486 DX Microprocessor D.C. Parametric Values (PGA Version)

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current CLK2 = 50 MHz		700	mA	(Note 3)
I_{LI}	Input Leakage Current		±15	μA	(Note 4)
I_{IH}	Input Leakage Current		200	μA	(Note 5)
I_{IL}	Input Leakage Current		−400	μA	(Note 6)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		20	pF	F_c = 1 MHz(7)
C_O	I/O or Output Capacitance		20	pF	F_c = 1 MHz(7)
C_{CLK}	CLK Capacitance		20	pF	F_c = 1 MHz(7)

NOTES:
1. This parameter is measured at:
   Address, Data BEn 4.0 mA
   Definition, Control 5.0 mA
2. This parameter is measured at:
   Address, Data BEn −1.0 mA
   Definition, Control −0.9 mA
3. Typical supply current
   I_{CC} = 550 mA @CLK2 = 50 MHz
4. This parameter is for inputs without pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with pulldowns and V_{IH} = 2.4V.
6. This parameter is for inputs with pullups and V_{IL} = 0.45V.
7. Not 100% tested.

15.8.2 POWER SUPPLY CURRENT vs FREQUENCY

Following is the power consumption of the Low Power Intel486 Microprocessor installed in a low power system for different frequencies.

Table 15.10. Power Supply Current (I_{CC}) Values over Frequencies of Operation (PGA Version)

CLK2 Frequency (MHz)	Operating Frequency (MHz)	$I_{CC(max)}$ (mA)	$I_{CC(typ)}$ (mA)
4	2	150	100
16	8	325	235
32	16	525	400
40	20	600	475
50	25	700	550

15.8.3 A.C. SPECIFICATIONS

The following table provides the A.C. specifications for the Low Power Intel486 DX Microprocessor. It consists of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the rising edge of the phase 1 of the input system clock (CLK2), unless otherwise specified.

Table 15.11. Low Power Intel486 DX—25 MHz Microprocessor A.C. Characteristics

V_{CC} = 5V ±10%; T_{CASE} = 0°C to +85°C; C_L = 50 pF(2) unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	2	25	MHz		Half of CLK2 Frequency
t_1	CLK2 Period	20	250	ns	15.8	
t_2	CLK2 High Time	7		ns	15.8	At 2V
t_3	CLK2 Low Time	7		ns	15.8	At 0.8V
t_4	CLK2 Fall Time		2	ns	15.8	2V to 0.8V
t_5	CLK2 Rise Time		2	ns	15.8	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	22	ns	15.9	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		30	ns	15.9	After Clock Edge(1)
t_8	PCHK# Valid Delay	3	27	ns	15.9	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	27	ns	15.10	

Table 15.11. Low Power Intel486 DX—25 MHz Microprocessor A.C. Characteristics (Continued)
V_{CC} = 5V ±10%; T_{CASE} = 0°C to +85°C; C_L = 50 pF(2) unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_9	BLAST#, PLOCK# Float Delay		30	ns	15.9	After Clock Edge(1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	22	ns	15.9	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		30	ns	15.9	After Clock Edge(1)
t_{12}	EADS# Setup Time	9		ns	15.10	
t_{13}	EADS# Hold Time	4		ns	15.10	
t_{14}	KEN#, BS16#, BS8# Setup Time	9		ns	15.10	
t_{15}	KEN#, BS16#, BS8# Hold Time	4		ns	15.10	
t_{16}	RDY#, BRDY# Setup Time	9		ns	15.10	
t_{17}	RDY#, BRDY# Hold Time	4		ns	15.10	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	11		ns	15.10	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	4		ns	15.10	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	11		ns	15.10	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	4		ns	15.10	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	6		ns	15.10	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	4		ns	15.10	
	CLKSEL	See Figures 15.4 and 15.5 for details on this signal. Figure 15.5 shows minimum timings required for the proper operation of the CPU. The pulse on CLKSEL can be of any length as long as the minimums are satisfied and the transitions from low to high occurs at the clock edge shown.				

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF.

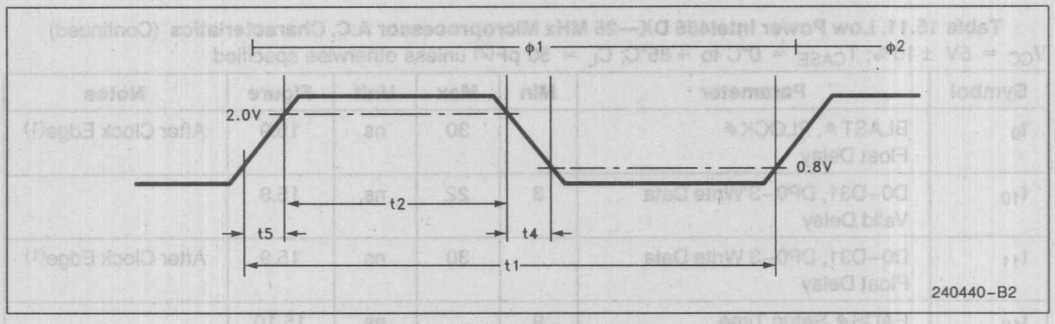


Figure 15.8. CLK2 Waveform

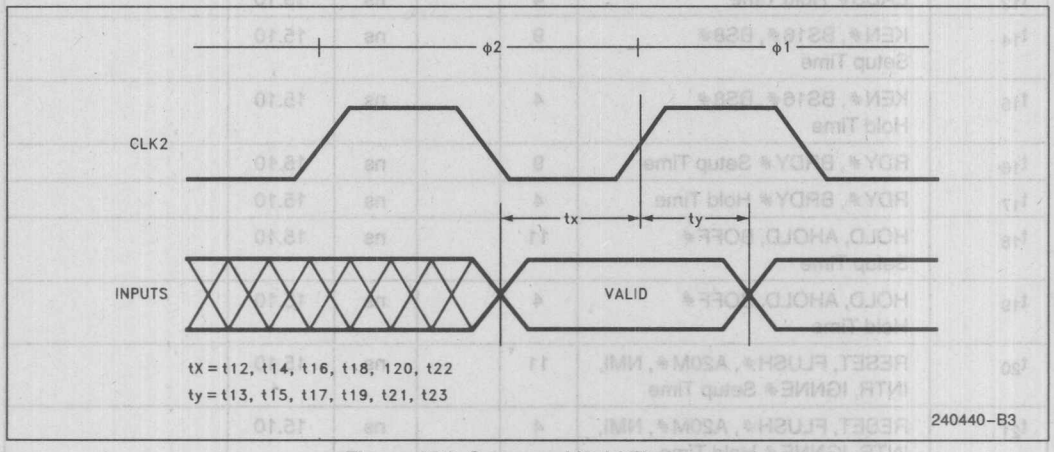


Figure 15.9. Setup and Hold Timings

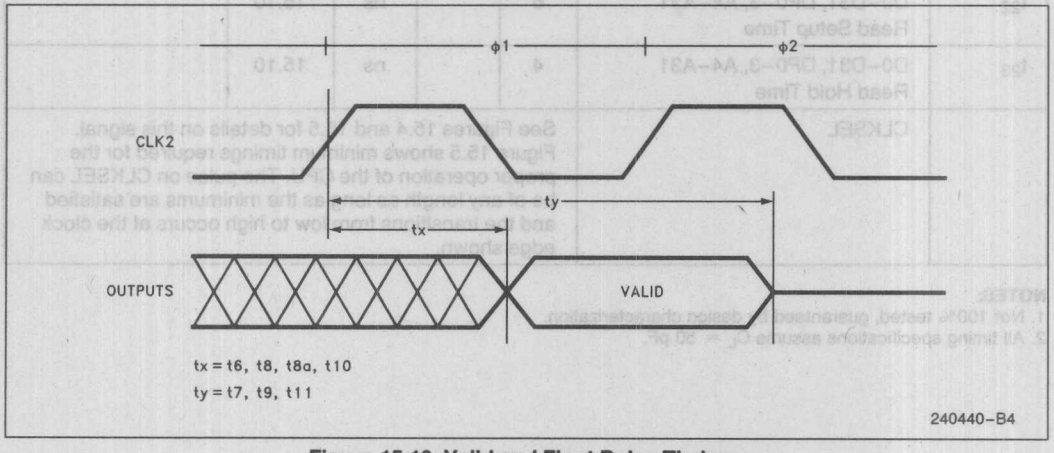


Figure 15.10. Valid and Float Delay Timings

16.0 SUGGESTED SOURCES FOR Intel486™ ACCESSORIES

Following are some suggested sources of accessories for the Intel486. They are not an endorsement of any kind, nor a warranty of the performance of any of the listed products and/or companies.

Sockets

1. McKenzie Technology
   44370 Old Palmspring Blvd.
   Fremont, CA 94538
   Tel: (415) 651-2700
2. E-CAM Technology, Inc.
   14455 North Hayden Rd.
   Suite 208
   Scottsdale, AZ 85260
   Tel: (602) 443-1949
3. Augat Inc. (for sockets with decaps)
   Interconnection Products Group
   33 Perry Ave.
   P.O. Box 779
   Attleboro, MA 02703
   Tel: (508) 222-2202

Heat Sinks/Fins

1. Thermalloy Inc.
   2021 West Valley View Lane
   Dallas, TX 75381-0839
   Tel: (214) 243-4321
2. E G & G Division
   60 Audubon Road
   Wakefield, MA 01880
   Tel: (617) 245-5900

TTL Crystals/Oscillators

1. NFL Frequency Controls, Inc.
   357 Beloit Street
   Burlington, WI 53105
   Tel: (414) 763-3591
2. M-Tron
   P.O. Box 630
   Yankton, SD 57078
   Tel: (605) 665-9321

Debugging Tower

1. Emulation Technology
   2344 Walsh Ave., Building F
   Santa Clara, CA 95051
   Tel: (408) 982-0664

17.0 REVISION HISTORY

Revision -006 of the Intel486 DX Microprocessor Data Book contains many updates and improvements to the original version. A revision summary of major changes is listed below:

The sections significantly revised since version -001 are:

Section 2.1.2 The polarity and names of the two cache control bits in Control Register 0 (CR0) have been modified. The Cache Enable (CE) and Writes Transparent (WR) have been renamed Cache Disable (CD) and Not Write Through (NW). The value of CR0 after RESET has been changed to reflect the polarity change.

Section 6.2.15 The discussion of A20M# has been clarifed. During the falling edge of RESET, A20M# should be high, for proper operation of the CPU.

Section 6.5 The value of CR0 after RESET has been modified.

Section 6.5.1 Figure 6.3, "Pin State during RESET" is added. This Figure is a general reference for Reset issues. Previous Figures 8.1, 8.2, and 8.8 have been deleted, since Figure 6.3 now contains Reset information.

Section 7.2.10 A discussion of addresses and byte enables driven during INTA cycles has been added.

Section 10.1 Clock counts and opcodes have been clarified and corrected.

Section 10.1 The opcode slot for CMPXCHG instruction has been moved from 0FA6/A7 to 0FB0/B1.

Section 12.2 Table 12.1 has been enhanced. The "Case Temperature under Bias" spec was improved. The "Supply Voltage with Respect to V_{SS}" spec was added.

Section 12.3 Maximum I_{CC} values have been improved to 700 mA at 25 MHz and 900 mA at 33 MHz.

Section 12.3 Typical I_{CC} values have been modified to 550 mA at 25 MHz and 700 mA at 33 MHz.

Section 12.3 C_{IN}, C_O, and C_{CLK} values have been changed to 20 pF. Testing parameters and Note 7 were added.

Section 12.4 The A.C. Specifications have been improved. Float delays were improved at both 25 MHz and 33 MHz. Note 1 was added to the float delays. Maximum valid delays were reduced at 33 MHz.

Section 12.5 The ICD section was enhanced.

Section 13.1 Thermal resistance θ_{CA} values of the 168-pin ceramic package have been corrected.

Section 13.1 Maximum ambient temperatures have been corrected to use the max I_{CC} values.

The sections significantly revised since version -002 are:

Section 2.1.2.1 Spec change for PC and PWT bits.

Table 2.16 Value of intel Reserved Interrupt Vector assignment corrected to '18-31'.

Section 3.1 Added CMPCHG, XADD instructions in the table.

Section 3.5 Added explanation about NMI not able to bring out the processor from shutdown under certain conditions.

Section 4.4.6 Value of task switching time corrected to 10 ms.

Section 4.5.4 Specification change for PCD and PWT bits.

Section 5.6 Specification change for PCD and PWT bits.

Section 5.7 Cache flushing procedure explained, when FLUSH# applied synchronously or asynchronously.

Section 6.2.5 Specification change for PLOCK cycle.

Section 6.2.8 Added explanation for warm boot-up.

Section 6.2.12 Specification change for PCD and PWT bits.

Section 6.2.13 Explanation added for FERR# behavior.

Section 6.2.14 Explanation added of IGNNE# behavior.

Section 6.2.15 Explanation added for A20M# behavior in protected mode and during RESET.

Section 6.3 Simplified example for read reordering in write buffers.

Section 6.3.1 Corrected REP OUTS instruction.

Section 6.3.2 Added explanation about cache update on read-modify-write cycle.

Section 6.5	Added RESET pulse length requirement with or without BIST
Section 6.5	Added table for Intel486 revision ID.
Table 6.2	Corrected CR0 value after Reset.
Figure 6.3	Corrected pin state diagram during RESET. RESET pulse length changed to 15 CLKs.
Section 7.2.2.3	Added explanation to terminate burst cycle.
Section 7.2.3.4	Clarified text on changing KEN# during cache line fill.
Figure 7.12	Corrected timing diagram to show A4–A31, M/IO#, D/C#, W/R# do not change during burst.
Figure 7.13	Corrected timing diagram to show A4–A31, M/IO#, D/C#, W/R# do not change during burst.
Figure 7.14	Corrected timing diagram to show A4–A31, M/IO#, D/C#, W/R# do not change during burst.
Section 7.2.4.2	Added cases that follow burst order.
Section 7.2.6	Added explanation for read-modify-write for un-aligned transfers.
Section 7.2.7	HOLD latency decreased by providing window in PLOCK cycle (specification change).
Section 7.2.8	Added explanation about EADS# timing.
Section 7.2.8	Added the case of invalidation with BOFF or HOLD.
Figure 7.22	Change in Timing Diagram for BREQ.
Figure 7.23	Change in Timing Diagram for BREQ.
Figure 7.25	Change in Timing Diagram for RDY#/BRDY#.
Section 7.2.9	Added explanation about HOLD getting recognized during unaligned writes.
Section 7.2.11	Added status of address and data busses during special bus cycles.
Section 7.2.11	Added sections on Halt and Shutdown cycles.
Figure 7.30	Corrected state diagram by ANDing BRDY# and BLAST# for the last transfer of the burst cycle.
Section 7.2.14	Difference in FERR# and ERROR# explained.
Section 8.1	Changed Reset width to 15 CLKs.
Section 8.4	Added explanation on tri-state status.
Table 10.1	Corrected value in format.
Section 11.0	Added Note 6 on FERR# and ERROR# difference.
Section 11.0	Added TLB replacement algorithm for 386 DX.
Section 12.3	Corrected values in Note 2.
Section 12.3	Added "internal" for pullup and pulldown resistors
Figure 12.2 & Figure 12.3	Waveforms for input and output signals have been re-drawn to show details about set-up, hold and float times.
Section 13.1	Added details about T_A calculation from θ_{JC} and θ_{JA}.
Section 14.0	Added new section on suggested sources of Intel486 accessories like sockets, debugging tower, heat sinks, etc.

The sections significantly revised since revision -003 are:

Cover Page	Add 50 MHz information to text and block diagram.
Figure 1.3 and Figure 1.4	Added 50 MHz pinout diagrams.
Pin Cross Reference Table	Added column for Test Access Port pins.
Quick Pin Reference	Added Test Access Port pin descriptions.
Table 1.4	Added Test Access Port pin sample/driven data.
Table 1.5	Added D0, cAx, and cBx revision ID information.
Section 6.2.9	Added description of HOLD recognition during BOFF#.
Section 6.2.12	Added PCD and PWT description when paging disabled.
Section 6.2.16	Added signal description for Test Access Port signals.
Table 6.3	Added D0, cAx, and cBx revision ID information.
Figure 6.4	Added additional details on signal samping during RESET.
Figure 7.30	Added HOLD to state transition between Tb and T1b.
Section 8.0, 8.5	Added Boundary Scan to test feature description.
Table 12.2	Added 50 MHz D.C. specifications.
Table 12.3	Added 50 MHz A.C. specifications.

Figure 12.7	Added Test signal timing reference diagram.
Figure 12.4.2	Added 50 MHz capacitive load derating curves.
Table 13.2	Added 50 MHz thermal resistance values.
Table 13.3	Added 50 MHz ambient temperature data.

The sections significantly revised since revision -004 are:

Intel OverDrive Processor information/specifications have been added throughout the document. Section 13.0 contains OverDrive Processor specific information.

Low Power Intel486 DX CPU information/specifications have been added. Section 15.0 contains Low Power specific information.

Quick Pin Reference	Clarified the description for KEN#.
Table 1-5	Updated component and revision I.D. information.
Section 6.2.15	More clearly defined A20M# bit by defining functionality during I/O writes, prefetching, etc.
Figure 6.4	Designated that FLUSH# must be inactive during BIST.
Section 7.2.15	Added section for floating point error handling in AT compatible systems.
Section 8.1	Clarified A20M#, FLUSH# and AHOLD functionality during BIST.
Section 8.5.7	BSDL is now available through Intel.
Table 10.1	Clarified CBW and CWD. Corrected REP LODS, REP MOVS and REP STOS.
Table 10.2	Corrected REP INS and REP OUTS.
Table 10.3	Corrected FSTP, FUCOMP, FSUBR, FDIV, FDIVR.
Appendix A	Added appendix for CPU Identification Code.

The sections significantly revised since -005 are:

Figure 1.5	Added PQFP package pinout.
PQFP pin tables	Added pin cross reference by signal type and complete pin reference tables.
Section 6.5	Added clarification for the built in self test (BIST) during reset.
Section 7.2.9	Added explanation of bus hold and hold acknowledge protocol.
Figure 7.26b	Added figure to illustrate HOLD request acknowledge during BOFF#.

APPENDIX A
INTEL RECOMMENDED CPU IDENTIFICATION CODE

The CPU identification assembly code will determine for the user which Intel microprocessor and if a Intel Math CoProcessor is installed in the system. If a 486 microprocessor has been installed, the program will determine if the CPU is with/without a floating point unit. This code should be executed so the system can be configured for a particular application, which may depend on the microprocessor and Math CoProcessor installed in the system.

```
        TITLE CPUID
        DOSSEG
        .model  small

        .stack  100h

        .data
fp_status       dw      ?
id_mess         db      "This system has a$"
fp_8087         db      "and an 8087 Math CoProcessor$"
fp_80287        db      "and an 287 Math CoProcessor$"
fp_80387        db      "and an 387 Math CoProcessor$"
c8086           db      "n8086/8088 microprocessor$"
c286            db      "n80286 microprocessor$"
c386            db      "386 microprocessor$"
c486            db      "486 DX microprocesser/487 SX Math
                        "CoProcessor$"
c486nfp         db      "486 SX Microprocesser$"
period          db      ".$",13,10
present_86      dw      0
present_286     dw      0
present_386     dw      0
present_486     dw      0
;
;       The purpose of this code is to allow the user the ability to identify
;       the processor and coprocesor that is currently in the system. The
;       algorithm of the program is to first determine the processor id.
;       When that is accomplished, the program continues to then identify
;       whether a coprocessor exists in the system. If a coprocessor or
;       integrated coprocessor exists, the program will identify the
;       coprocessor id. If one does not exist, the program then terminates.
;

        .code
start:
                mov     ax,@data
                mov     ds,ax                   ; set segment register

                mov     dx,offset id_mess       ;print header message
                mov     ah,9h
                int     21h
```

```
;
;       8086 check
;       Bits 12-15 are always set on the 8086 processor.
;
                pushf                           ; save EFLAGS
                pop     bx                      ; store EFLAGS in BX
                mov     ax,0fffh                ; clear bits 12-15
                and     ax,bx                   ;       in EFLAGS
                push    ax                      ; store new EFLAGS value on stack
                popf                            ; replace current EFLAGS value
                pushf                           ; set new EFLAGS
                pop     ax                      ; store new EFLAGS in AX
                and     ax,0f000h               ; if bits 12-15 are set, then CPU
                cmp     ax,0f000h               ;       is an 8086/8088
                mov     dx,offset c8086         ; store 8086/8088 message
                mov     present_86,1            ; turn on 8086/8088 flag
                je      check_fpu               ; if CPU is 8086/8088, check for
                                                ; 8087

;
;       80286 CPU Check
;       Bits 12-15 are always clear on the 80286 processor.
;

                or      bx,0f000h               ; try to set bits 12-15
                push    bx
                popf
                pushf
                pop     ax
                and     ax,0f000h               ; if bits 12-15 are cleared, then
                mov     dx,offset c286          ;       CPU is an 80286
                mov     present_86,0            ; turn off 8086/8088 flag
                mov     present_286,1           ; turn on 80286 flag
                jz      check_fpu               ; if CPU is 80286, check for 80287

;
;       386 CPU check
;       The AC bit, bit #18, is a new bit introduced in the EFLAGS register
;       on the 486 DX CPU to generate alignment faults. This bit can be set
;       on the 486 DX CPU, but not on the 386 CPU.
;

                mov     bx,sp                   ; save current stack pointer to
                                                ; align it
                and     sp,not 3                ; align stack to avoid AC fault
                db      66h
                pushf                           ; push original EFLAGS
                db      66h
                pop     ax                      ; get original EFLAGS
                db      66h
                mov     cx,ax                   ; save original EFLAGS
                db      66h                     ; xor EAX,40000h
                xor     ax,0                    ; flip AC bit in EFLAGS
                dw      4                       ; upper 16-bits of xor constant
                db      66h
                push    ax                      ; save for EFLAGS
                db      66h
                popf                            ; copy to EFLAGS
```

```
                db      66h
                pushf                   ; push EFLAGS
                db      66h
                pop     ax              ; get new EFLAGS value
                db      66h
                xor     ax,cx           ; if AC bit cannot be changed,
                                        ; CPU is
                mov     dx,offset c386  ; store 386 message
                mov     present_286,0   ; turn off 80286 flag
                mov     present_386,1   ; turn on 386 flag
                je      check_fpu       ; if CPU is 386, now check for
                                        ;  80287/80387
;
;       486 DX CPU and 486 DX CPU w/o FPU checking
;
                mov     dx,offset c486nfp   ; store 486NFP message
                mov     present_386,0   ; turn off 386 flag
                mov     present_486,1   ; turn on 486 flag
;
;       Co-processor checking begins here for the 8086/80286/386 CPUs.
;       The algorithm is to determine whether or not the floating-point
;       status and control words can be written to, the correct coprocessor
;       is then determined depending on the processer id. Coprocessor checks
;       are first performed for an 8086, 80286 and a 486 DX CPU. If the
;       coprocessor id is still undetermined, the system must contain a 386
;       CPU. The 386 CPU may work with either an 80287 or an 80387. The
;       infinity of the coprocessor must be checked to determine the correct
;       coprocessor id.
;
check_fpu:                              ; check for 8087/80287/80387
                fninit                  ; reset FP status word
                mov     fp_status,5a5ah ; initialize temp word to non-zero
                                        ; value
                fnstsw fp_status        ; save FP status word
                mov     ax,fp_status    ; check FP status word
                cmp     al,0            ; see if correct status with
                                        ; written
                jne     print_one       ; jump if not Valid, no NPX
                                        ; installed
                fnstcw fp_status        ; save FP control word
                mov     ax,fp_status    ; check FP control word
                and     ax,103fh        ; see if selected parts looks OK
                cmp     ax,3fh          ; check that ones and zeroes
                                        ; correctly read
                jne     print_one       ; jump if not Valid, no NPX
                                        ; installed

                cmp     present_486,1   ; check if 486 flag is on
                je      is_486          ; if so, jump to print 486 message
                jmp     not_486         ; else continue with 386 checking

is_486:
                mov     dx,offset c486  ; store 486 message
                jmp     print_one
```

```
not_486:
                cmp     present_386,1           ; check if 386 flag is on
                jne     print_87_287            ; if 386 flag not on, check NPX for
                                                ;  8086/8088/80286
                mov     ah,9h                   ; print out 386 CPU ID first
                int     21h
;
;       80287/80387 check for the 386 CPU
;
                fldl                            ; must use default control from
                                                ; FNINIT
                fldz                            ; form infinity
                fdiv                            ; 8087/80287 says +inf = inf
                fld     st                      ; form negative infinity
                fchs                            ; 80387 says +inf <> -inf
                fcompp                          ; see if they are the same and
                                                ; remove them
                fstsw   fp_status               ; look at status from FCOMPP
                mov     ax,fp_status
                mov     dx,offset fp_80287      ; store 80287 message
                sahf                            ; see if infinities matched
                jz      restore_EFLAGS          ; jump if 8087/80287 is present
                mov     dx,offset fp_80387      ; store 80387 message
restore_EFLAGS:
                finit                           ; clear any pending fp exception
                mov     ah,9h                   ; print NPX message
                int     21h
                db      66h
                push    cx                      ; push ECX
                db      66h
                popf                            ; restore original EFLAGS register
                mov     sp,bx                   ; restore original stack pointer
                jmp     exit
print_one:
                mov     ah,9h                   ; print out CPU ID with no NPX
                int     21h
                jmp     exit
print_87_287:
                mov     ah,9h                   ; print out 8086/8088/80286 first
                int     21h
                cmp     present_86,1            ; if 8086/8088 flag is on
                mov     dx,offset fp_8087       ; store 8087 message
                je      print_fpu
                mov     dx,offset fp_80287      ; else CPU = 80286, store 80287
                                                ; message
print_fpu:
                mov     ah,9h                   ; print out NPX
                int     21h
                jmp     exit
exit:
                mov     dx,offset period        ; print out a period of end message
                mov     ah,9h
                int     21h

                mov     ax,4c00h                ; terminate program
                int     21h

                end     start
```

Intel486™ SX MICROPROCESSOR

IMPORTANT—Read This Section Before Reading The Rest Of The Data Sheet

This data sheet describes the Intel486 SX microprocessor, the Intel OverDrive™ Processor, and the Intel487™ SX Math CoProcessor. All normal text describes the functionality for the Intel486 SX microprocessor, the Intel OverDrive Processor, and the Intel487 SX Math CoProcessor unless explicitly stated otherwise. All sections of the data sheet which describe the Intel487 SX Math CoProcessor and Intel OverDrive Processor functionality only are highlighted as shown in the following example:

> This is an example of what the highlighted sections look like. The highlighted sections describe functionality which apply to the Intel487 SX Math CoProcessor and Intel OverDrive Processor only.

All references to the Intel OverDrive Processor are also applicable to the Intel487 SX Math CoProcessor unless otherwise stated.

Section 13.0, OverDrive Processor, contains additional information specific to the OverDrive Processor.

2

- **Binary Compatible with Large Software Base**
  - **MS-DOS*, OS/2**, Windows**
  - **UNIX*** System V/386**
  - **iRMX®, iRMK Kernels**
- **High Integration Enables On-Chip**
  - **8 Kbyte Code and Data Cache**
  - **Paged, Virtual Memory Management**
- **Easy To Use**
  - **Built-In Self Test**
  - **Hardware Debugging Support**
  - **Intel Software Support**
  - **Extensive Third Party Software Support**
- **196-Lead PQFP and 168-Pin Grid Array Package for Intel486™ SX Microprocessor**
- **169-Pin Grid Array Package for Intel OverDrive™ Processor and Intel487™ SX Math CoProcessor**
- **High Performance Design**
  - **Intel486 One Clock Instruction Core**
  - **33, 25, 20 and 16 MHz Clock Frequencies at 5V**
  - **25, 20 and 16 MHz Clock Frequencies at 3.3V**
  - **106 Mbyte/sec Burst Bus at 33 MHz**
  - **CHMOS IV and CHMOS V Process Technology**
  - **Dynamic Bus Sizing for 8-, 16- and 32-Bit Busses**
- **Complete 32-Bit Architecture**
  - **Address and Data Busses**
  - **Registers**
  - **8-, 16- and 32-Bit Data Types**
- **Multiprocessor Support**
  - **Multiprocessor Instructions**
  - **Cache Consistency Protocols**
  - **Support for Second Level Cache**
- **Optional Intel OverDrive™ Processor/ Intel487™ SX Math CoProcessor (16–25 MHz only)**
- **IEEE 1149.1 Boundary Scan Compatibility**
  - **Available on PQFP Intel486 SX CPU Only**

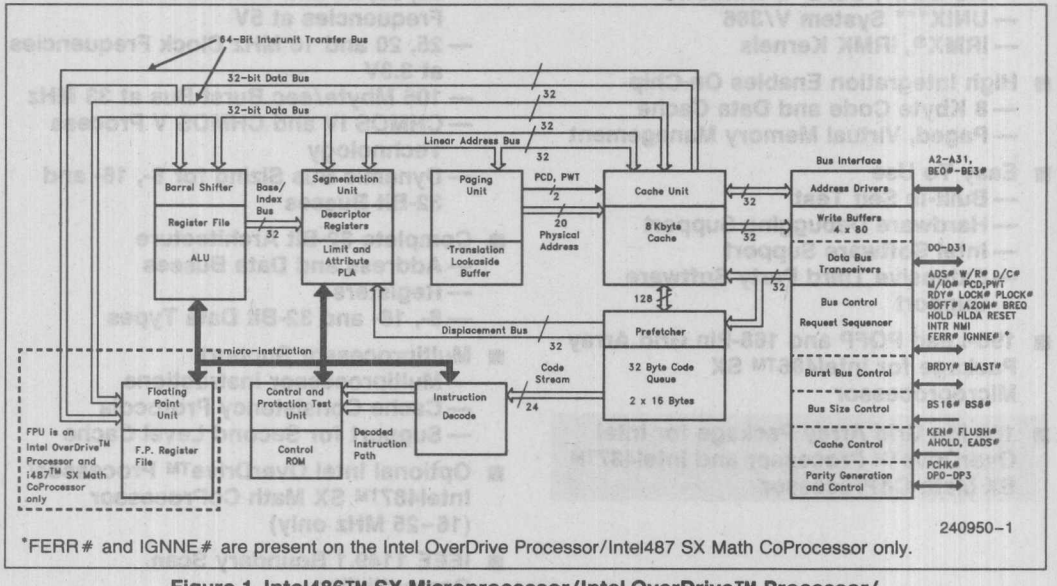
Intel386™, Intel387™, i386™, i387™, Intel486™, Intel487™, i486™, OverDrive™ and i487™ are trademarks of Intel Corporation.
*MS-DOS, WINDOWS are registered trademarks of Microsoft Corporation.
**OS/2 is a trademark of Microsoft Corporation.
***UNIX is a trademark of Unix Systems Labs.

November 1993
Order Number: 240950-004

Intel486™ SX CPU based systems are the entry-level business standard, delivering affordable performance and upgradability with Intel's OverDrive™ Processor. The Intel486 SX CPU is the industry standard for entry level business computing. It has replaced the 80386 as the minimum performance requirement for the demands of today's software. Intel486 SX CPU based systems are now available at comparable prices to 80386 systems. Additionally, they offer investment protection through built-in single chip upgradability with Intel's OverDrive Processor. The Intel486 SX microprocessor is a binary compatible member of the Intel486 microprocessor family, giving it access to the $50 Billion installed software base of over 50,000 MS-DOS, Windows, OS/2, and UNIX System V/386 applications. The Intel486 SX microprocessor has the same integrated RISC integer core, 8 KByte cache memory, and memory management unit as the Intel486 DX microprocessor. The Intel OverDrive Processor provides optional overall performance upgrade capability for users who want to increase their system performance up to 70% on DOS, UNIX, WINDOWS, OS/2, and UNIX applications.

The high-performance RISC integer core of the Intel486 SX microprocessor executes frequently used instructions in one clock cycle. An 8 KByte unified code and data cache allow this performance level to be sustained. A 106 Mbyte/sec burst bus at 33 MHz ensures high system throughput even with inexpensive DRAMs. The 33 MHz Intel486 SX microprocessor has a Norton SI V5.0 rating of 72.1, a Dhrystone MIPS rating of 27, and a 9.5 SPECint92 rating. It provides up to twice the performance of a 33 MHz Intel386 DX microprocessor with an external cache (depending on the application).

NOTE:
The Intel486 SX CPU and Intel OverDrive Processor are available at 33 MHz external clock frequency. However, the Intel487 SX Math CoProcessor is not available at 33 MHz.



*FERR# and IGNNE# are present on the Intel OverDrive Processor/Intel487 SX Math CoProcessor only.

Figure 1. Intel486™ SX Microprocessor/Intel OverDrive™ Processor/ Intel487™ SX Math CoProcessor Pipelined 32-Bit Microarchitecture

Intel486™ SX Microprocessor

CONTENTS

PAGE

1.0 TABLE OF CONTENTS 2-447
Pinout 2-451
Quick Pin Reference 2-460
Component and Revision ID 2-466
2.0 ARCHITECTURAL OVERVIEW 2-467
2.0.1 PQFP Package 2-468
2.1 Register Set 2-468
2.1.1 Base Architecture Registers 2-469
2.1.2 System Level Registers 2-473
2.1.3 Floating Point Registers 2-479
2.1.4 Debug and Test Registers .. 2-486
2.1.5 Register Accessibility 2-486
2.1.6 Compatibility 2-487
2.2 Instruction Set 2-488
2.3 Memory Organization 2-488
2.3.1 Address Spaces 2-488
2.3.2 Segment Register Usage ... 2-489
2.4 I/O Space 2-489
2.5 Addressing Modes 2-490
2.5.1 Addressing Modes Overview 2-490
2.5.2 Register and Immediate Modes 2-490
2.5.3 32-Bit Memory Addressing Modes 2-490
2.5.4 Differences between 16- and 32-Bit Addresses 2-492
2.6 Data Formats 2-492
2.6.1 Data Types 2-492
2.6.2 Little Endian vs Big Endian Data Formats 2-496
2.7 Interrupts 2-496
2.7.1 Interrupts and Exceptions ... 2-496
2.7.2 Interrupt Processing 2-496
2.7.3 Maskable Interrupt 2-497
2.7.4 Non-Maskable Interrupt 2-498
2.7.5 Software Interrupts 2-498
2.7.6 Interrupt and Exception Priorities 2-498
2.7.7 Instruction Restart 2-499
2.7.8 Double Fault 2-499

CONTENTS

PAGE

2.7.9 Floating Point Interrupt Vectors 2-499
3.0 REAL MODE ARCHITECTURE 2-501
3.1 Real Mode Introduction 2-501
3.2 Memory Addressing 2-501
3.3 Reserved Locations 2-502
3.4 Interrupts 2-502
3.5 Shutdown and Halt 2-502
4.0 PROTECTED MODE ARCHITECTURE 2-503
4.1 Introduction 2-503
4.2 Addressing Mechanism 2-503
4.3 Segmentation 2-504
4.3.1 Segmentation Introduction .. 2-504
4.3.2 Terminology 2-504
4.3.3 Descriptor Tables 2-504
4.3.4 Descriptors 2-506
4.4 Protection 2-515
4.4.1 Protection Concepts 2-515
4.4.2 Rules of Privilege 2-516
4.4.3 Privilege Levels 2-516
4.4.4 Privilege Level Transfers 2-517
4.4.5 Call Gates 2-520
4.4.6 Task Switching 2-520
4.4.7 Initialization and Transition to Protected Mode 2-521
4.4.8 Tools for Building Protected Systems 2-522
4.5 Paging 2-522
4.5.1 Paging Concepts 2-522
4.5.2 Paging Organization 2-523
4.5.3 Page Level Protection (R/W, U/S Bits) 2-524
4.5.4 Page Cacheability (PWT, PCD Bits) 2-525
4.5.5 Translation Lookaside Buffer 2-525
4.5.6 Paging Operation 2-526
4.5.7 Operating System Responsibilities 2-527
4.6 Virtual 8086 Environment 2-527

CONTENTS

PAGE

4.6.1 Executing 8086 Programs ... 2-527
4.6.2 Virtual 8086 Addressing Mechanism ... 2-527
4.6.3 Paging in Virtual Mode ... 2-527
4.6.4 Protection and Virtual 8086 Mode to I/O Permission Bitmap ... 2-528
4.6.5 Interrupt Handling ... 2-529
4.6.6 Entering and Leaving Virtual 8086 Mode ... 2-530

5.0 ON-CHIP CACHE ... 2-533
5.1 Cache Organization ... 2-533
5.2 Cache Control ... 2-534
5.3 Cache Line Fills ... 2-534
5.4 Cache Line Invalidations ... 2-535
5.5 Cache Replacement ... 2-535
5.6 Page Cacheability ... 2-536
5.7 Cache Flushing ... 2-537
5.8 Caching Translation Lookaside Buffer Entries ... 2-537

6.0 HARDWARE INTERFACE ... 2-538
6.1 Introduction ... 2-538
6.2 Signal Descriptions ... 2-539
6.2.1 Clock (CLK) ... 2-539
6.2.2 Address Bus (A31–A2, BE0#–BE3#) ... 2-539
6.2.3 Data Lines (D31–D0) ... 2-539
6.2.4 Parity ... 2-540
Data Parity Input/Outputs (DP0–DP3) ... 2-540
Parity Status Output (PCHK#) ... 2-540
6.2.5 Bus Cycle Definition ... 2-540
M/IO#, D/C#, W/R# Outputs ... 2-540
Bus Lock Output (LOCK#) ... 2-540
Pseudo-Lock Output (PLOCK#) ... 2-541
6.2.6 Bus Control ... 2-541
Address Status Output (ADS#) ... 2-541
Non-Burst Ready Input (RDY#) ... 2-541
6.2.7 Burst Control ... 2-541

CONTENTS

PAGE

Burst Ready Input (BRDY#) ... 2-541
Burst Last Output (BLAST#) ... 2-542
6.2.8 Interrupt Signals ... 2-542
Reset Input (RESET) ... 2-542
Maskable Interrupt Request Input (INTR) ... 2-542
Non-Maskable Interrupt Request Input (NMI) ... 2-542
6.2.9 Bus Arbitration Signals ... 2-542
Bus Request Output (BREQ) ... 2-542
Bus Hold Request Input (HOLD) ... 2-542
Bus Hold Acknowledge Output (HLDA) ... 2-543
Backoff Input (BOFF#) ... 2-543
6.2.10 Cache Invalidation ... 2-543
Address Hold Request Input (AHOLD) ... 2-543
External Address Valid Input (EADS#) ... 2-544
6.2.11 Cache Control ... 2-544
Cache Enable Input (KEN#) ... 2-544
Cache Flush Input (FLUSH#) ... 2-544
6.2.12 Page Cacheability Outputs (PWT, PCD) ... 2-544
6.2.13.1 Intel487 SX Math CoProcessor Signals ... 2-544
6.2.13.2 Numeric Error Reporting ... 2-545
Floating Point Error Output (FERR#) ... 2-545
Ignore Numeric Error Input (IGNNE#) ... 2-545
6.2.14 Bus Size Control (BS16#, BS8#) ... 2-545
6.2.15 Address Bit 20 Mask (A20M#) ... 2-545
6.2.16 Performance Upgrade Support Signal Description ... 2-546
6.2.17 Boundary Scan Test Signals ... 2-546
6.3 Write Buffers ... 2-547
6.3.1 Write Buffers and I/O Cycles ... 2-547

CONTENTS

PAGE

6.3.2 Write Buffers Implications on Locked Bus Cycles 2-548
6.4 Interrupt and Non-Maskable Interrupt Interface 2-548
6.4.1 Interrupt Logic 2-548
6.4.2 NMI Logic 2-548
6.5 Reset and Initialization 2-549
6.5.1 Pin State during Reset 2-550
6.6 Math Upgrade Circuit 2-553
6.6.1 Upgrade Circuit for Intel486 SX CPU in PGA 2-553
6.6.2 Upgrade Circuit for Intel486 SX CPU in PQFP 2-553
6.6.3 Design Considerations 2-553

7.0 BUS OPERATION 2-557
7.1 Data Transfer Mechanism 2-557
7.1.1 Memory and I/O Spaces 2-557
7.1.2 Memory and I/O Space Organization 2-558
7.1.3 Dynamic Data Bus Sizing 2-559
7.1.4 Interfacing with 8-, 16- and 32-bit Memories 2-560
7.1.5 Dynamic Bus Sizing during Cache Line Fills 2-562
7.1.6 Operand Alignment 2-562
7.2 Bus Functional Description 2-563
7.2.1 Non-Cacheable Non-Burst Single Cycle 2-563
7.2.2 Multiple and Burst Cycle Bus Transfers 2-564
7.2.3 Cacheable Cycles 2-568
7.2.4 Burst Mode Details 2-571
7.2.5 8- and 16-Bit Cycles 2-575
7.2.6 Locked Cycles 2-577
7.2.7 Pseudo-Locked Cycles 2-578
7.2.8 Invalidate Cycles 2-578
7.2.9 Bus Hold 2-582
7.2.10 Interrupt Acknowledge 2-583
7.2.11 Special Bus Cycles 2-585
7.2.12 Bus Cycle Restart 2-586
7.2.13 Bus States 2-587
7.2.14 Floating Point Error Handling 2-588
7.2.15 Floating Point Error Handling in AT Compatible Systems 2-588

CONTENTS

PAGE

8.0 TESTABILITY 2-591
8.1 Built-In Self Test (BIST) 2-591
8.2 On-Chip Cache Testing 2-591
8.2.1 Cache Testing Registers TR3, TR4 and TR5 2-591
Cache Data Test Register: TR3 2-592
Cache Status Test Register: TR4 2-592
Cache Control Test Register: TR5 2-592
8.2.2 Cache Testability Write 2-593
8.2.3 Cache Testability Read 2-595
8.2.4 Flush Cache 2-595
8.3 Translation Lookaside Buffer (TLB) Testing 2-595
8.3.1 Translation Lookaside Buffer Organization 2-595
8.3.2 TLB Test Registers: TR6 and TR7 2-596
Command Test Register: TR6 2-597
Data Test Register: TR7 2-597
8.3.3 TLB Write Test 2-598
8.3.4 TLB Lookup Test 2-598
8.4 Tristate Output Test Mode 2-598
8.5 Intel486™ SX CPU Microprocessor Boundary Scan (JTAG) 2-599
8.5.1 Boundary Scan Architecture 2-599
8.5.2 Data Registers 2-599
8.5.3 Instruction Register 2-600
8.5.4 Test Access Port (TAP) Controller 2-602
8.5.5 Boundary Scan Register Cell 2-604
8.5.6 Tap Controller Initialization 2-605
8.5.7 Boundary Scan Description Language (BSDL) 2-605

9.0 DEBUGGING SUPPORT 2-606
9.1 Breakpoint Instructions 2-606
9.2 Single Step Instructions 2-606
9.3 Debug Registers 2-606
9.3.1 Linear Address Breakpoint Registers 2-606
9.3.2 Debug Control Register 2-606

CONTENTS

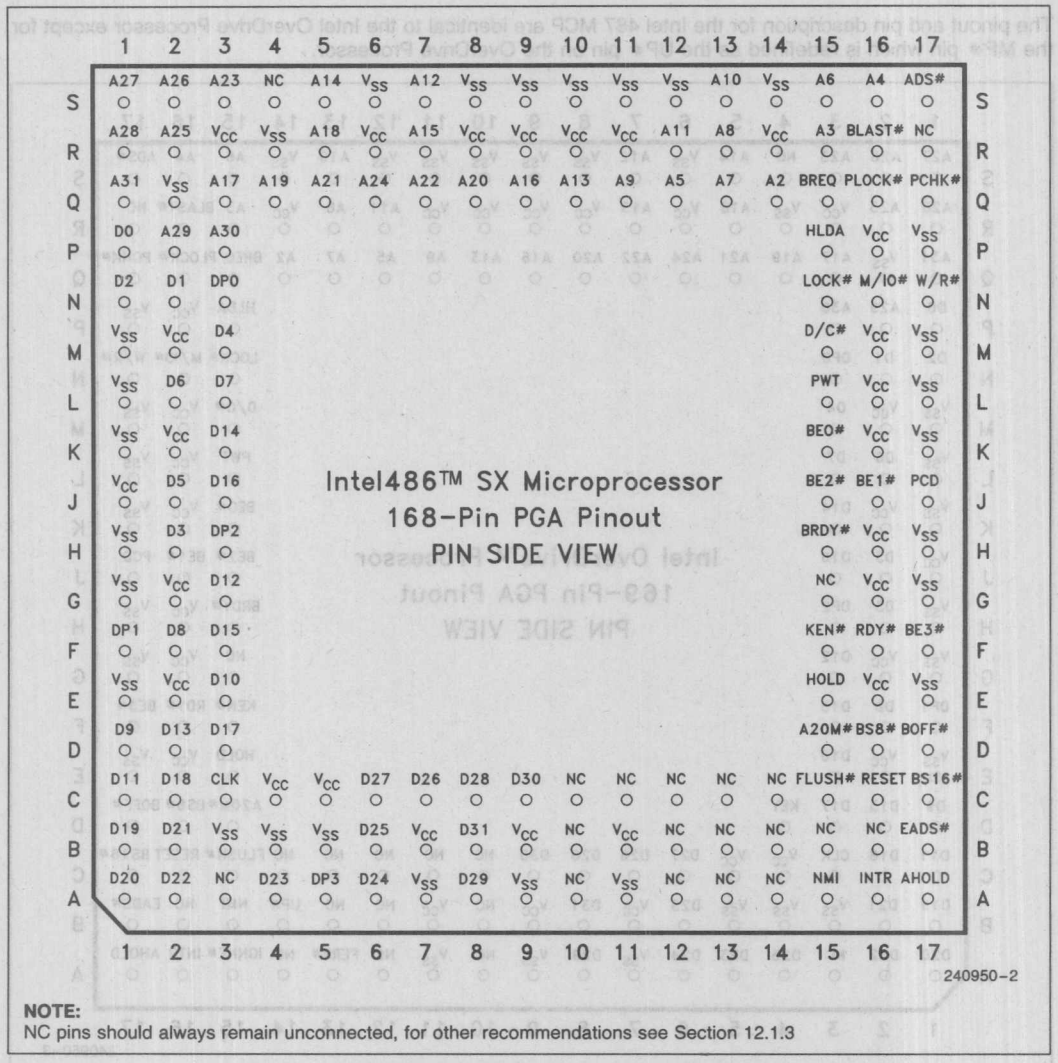
PAGE

9.3.3 Debug Status Register 2-609
9.3.4 Use of Resume Flag (RF) in Flag Register 2-610
10.0 INSTRUCTION SET SUMMARY .. 2-611
10.1 Intel486 SX Microprocessor/ Intel487 SX Math CoProcessor Instruction Encoding and Clock Count Summary 2-611
10.2 Instruction Encoding 2-630
10.2.1 Overview 2-630
10.2.2 32-Bit Extensions of the Instruction Set 2-631
10.2.3 Encoding of Integer Instruction Fields 2-632
10.2.4 Encoding of Floating Point Instruction Fields 2-638
11.0 DIFFERENCES WITH THE Intel386™ MICROPROCESSOR 2-639
12.0 DIFFERENCES WITH THE Intel486™ SX MICROPROCESSOR IN PGA vs. PQFP 2-640
13.0 OverDrive™ PROCESSOR 2-641
13.1 Hardware Interface 2-641
13.2 Testability 2-641
13.3 Instruction Set Summary 2-641
13.4 Bios and Software 2-642
13.4.1 Intel OverDrive Processor Detection 2-642
13.4.2 Timing Dependent Loops .. 2-643
13.5 Thermal Management 2-643
13.5.1 The Intel OverDrive Processor with Attached Heat Sink 2-643
13.5.2 Intel OverDrive Processor without Heat Sink 2-644
13.5.3 Thermal Equations 2-644
14.0 ELECTRICAL DATA 2-645
14.1 Power and Grounding 2-645
14.1.1 Power Connections 2-645
14.1.2 Power Decoupling Recommendations 2-645
14.1.3 Other Connection Recommendations 2-645
14.2 Maximum Ratings 2-645
14.3 D.C. Specifications 2-646
14.4 A.C. Specifications 2-649
14.5 Designing for ICD-486 2-657

CONTENTS

PAGE

15.0 MECHANICAL DATA 2-662
15.0.1 Intel486 SX Microprocessor 168 Lead Ceramic PGA Package 2-662
15.0.2 Intel486 SX Microprocessor 196 Lead PQFP Package 2-664
15.0.3 Intel OverDrive Processor 169 Lead PGA Package 2-666
15.1 Package Thermal Specifications 2-668
15.1.1 Intel486 SX Microprocessor PGA Package 2-670
15.1.2 Intel486 SX Microprocessor PQFP Package 2-671
15.1.3 Intel OverDrive Processor 2-672
16.0 LOW POWER Intel486™ SX MICROPROCESSOR 2-673
16.1 Introduction 2-673
16.2 Pinout 2-675
16.3 Pin Cross Reference (Intel486 SX CPU) 2-678
16.4 Pin Cross Reference (Intel486 SX CPU PQFP Version) 2-679
16.5 Pin Description 2-680
16.6 Signal Description 2-681
16.7 Architecture Overview 2-683
16.8 Variable CPU Frequency 2-684
16.9 D.C./A.C. Specifications 2-684
16.9.1 D.C. Specifications 2-685
16.9.2 Power Supply Current vs Frequency 2-687
16.9.3 A.C. Specifications 2-687
16.10 Match Upgrade for Low Power Intel486 SX CPU 2-692
16.10.1 Pinout 2-693
16.10.2 Pin Reference of Intel487 SX Match CoProcessor 2-695
16.10.3 Intel487 SX Math CoProcessor Pin Description ... 2-696
17.0 SUGGESTED SOURCES FOR Intel486™ SX MICROPROCESSOR/ Intel OverDrive™ PROCESSOR ACCESSORIES 2-697
18.0 REVISION HISTORY 2-698
APPENDIX A 2-701
APPENDIX B 2-703
APPENDIX C: 3.3V Intel486™ SX MICROPROCESSOR 2-707



NOTE:
NC pins should always remain unconnected, for other recommendations see Section 12.1.3

Figure 1.1a. Intel486™ SX Microprocessor Pin Side View

The pinout and pin description for the Intel 487 MCP are identical to the Intel OverDrive Processor except for the MP# pin which is redefined as the UP# pin on the OverDrive Processor.

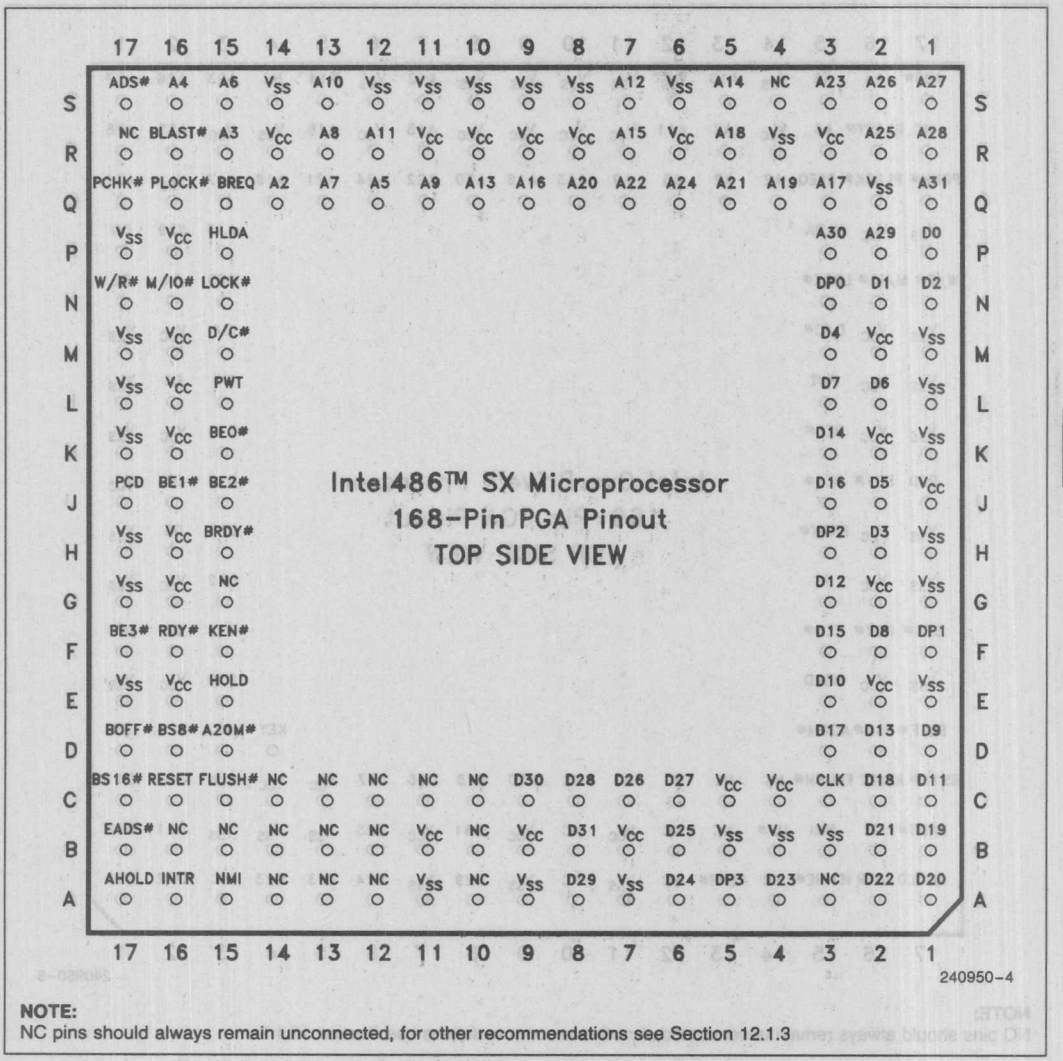
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
S	A27	A26	A23	NC	A14	V_{SS}	A12	V_{SS}	V_{SS}	V_{SS}	V_{SS}	V_{SS}	A10	V_{SS}	A6	A4	ADS#	S
R	A28	A25	V_{CC}	V_{SS}	A18	V_{CC}	A15	V_{CC}	V_{CC}	V_{CC}	V_{CC}	A11	A8	V_{CC}	A3	BLAST#	NC	R
Q	A31	V_{SS}	A17	A19	A21	A24	A22	A20	A16	A13	A9	A5	A7	A2	BREQ	PLOCK#	PCHK#	Q
P	D0	A29	A30												HLDA	V_{CC}	V_{SS}	P
N	D2	D1	DP0												LOCK#	M/IO#	W/R#	N
M	V_{SS}	V_{CC}	D4												D/C#	V_{CC}	V_{SS}	M
L	V_{SS}	D6	D7												PWT	V_{CC}	V_{SS}	L
K	V_{SS}	V_{CC}	D14												BEO#	V_{CC}	V_{SS}	K
J	V_{CC}	D5	D16												BE2#	BE1#	PCD	J
H	V_{SS}	D3	DP2												BRDY#	V_{CC}	V_{SS}	H
G	V_{SS}	V_{CC}	D12												NC	V_{CC}	V_{SS}	G
F	DP1	D8	D15												KEN#	RDY#	BE3#	F
E	V_{SS}	V_{CC}	D10												HOLD	V_{CC}	V_{SS}	E
D	D9	D13	D17	KEY											A20M#	BS8#	BOFF#	D
C	D11	D18	CLK	V_{CC}	V_{CC}	D27	D26	D28	D30	NC	NC	NC	NC	NC	FLUSH#	RESET	BS16#	C
B	D19	D21	V_{SS}	V_{SS}	V_{SS}	D25	V_{CC}	D31	V_{CC}	NC	V_{CC}	NC	NC	UP#	NMI	NC	EADS#	B
A	D20	D22	NC	D23	DP3	D24	V_{SS}	D29	V_{SS}	NC	V_{SS}	NC	FERR#	NC	IGNNE#	INTR	AHOLD	A
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	

Intel OverDrive™ Processor
169-Pin PGA Pinout
PIN SIDE VIEW

240950-3

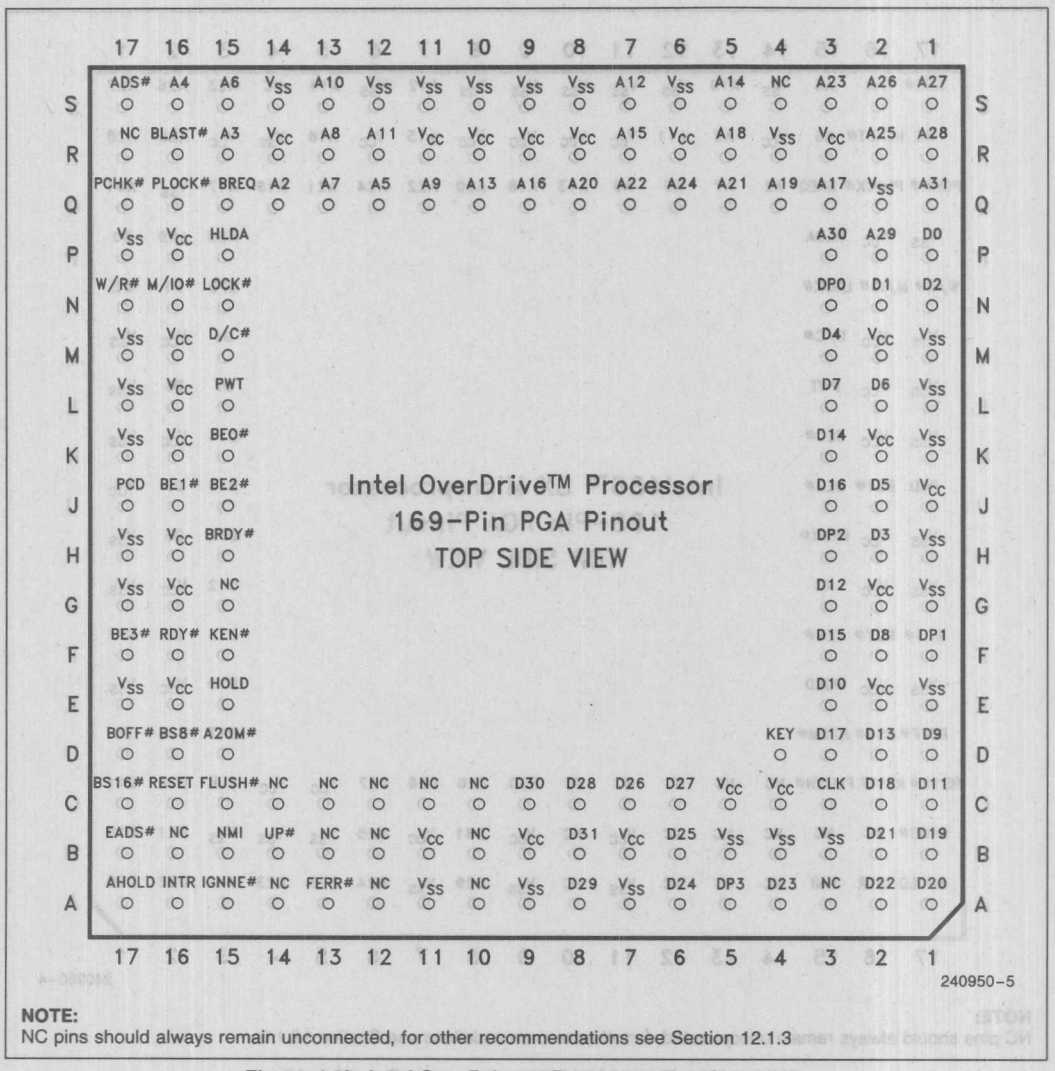
NOTE:
NC pins should always remain unconnected, for other recommendations see Section 12.1.3

Figure 1.1b. Intel OverDrive™ Processor Pinout Pin Side View



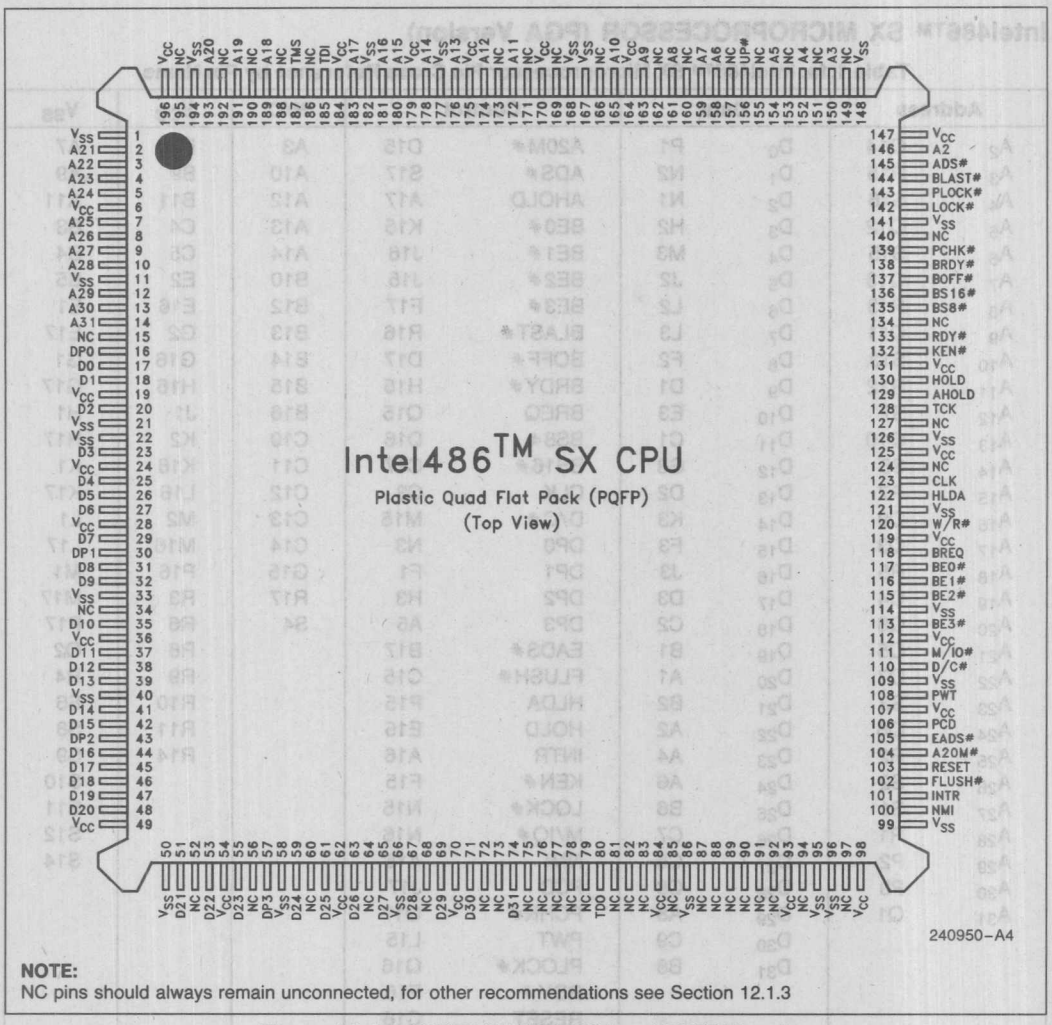
NOTE:
NC pins should always remain unconnected, for other recommendations see Section 12.1.3

Figure 1.2a. Intel486™ Microprocessor Top Side View



NOTE:
NC pins should always remain unconnected, for other recommendations see Section 12.1.3

Figure 1.2b. Intel OverDrive™ Processor Top Side View



NOTE:
NC pins should always remain unconnected, for other recommendations see Section 12.1.3

Figure 1.3. Intel486™ SX CPU 196 Lead PQFP Pinout

Table 1.1a. Intel486™ SX Microprocessor Pin Cross Reference by Pin Name

Address		Data		Control		N/C	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	A3	B7	A7
A_3	R15	D_1	N2	ADS#	S17	A10	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	A12	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	A13	C4	B3
A_6	S15	D_4	M3	BE1#	J16	A14	C5	B4
A_7	Q13	D_5	J2	BE2#	J15	B10	E2	B5
A_8	R13	D_6	L2	BE3#	F17	B12	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16	B13	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17	B14	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15	B15	H16	G17
A_{12}	S7	D_{10}	E3	BREQ	Q15	B16	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16	C10	K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17	C11	K16	K1
A_{15}	R7	D_{13}	D2	CLK	C3	C12	L16	K17
A_{16}	Q9	D_{14}	K3	D/C#	M15	C13	M2	L1
A_{17}	Q3	D_{15}	F3	DP0	N3	C14	M16	L17
A_{18}	R5	D_{16}	J3	DP1	F1	G15	P16	M1
A_{19}	Q4	D_{17}	D3	DP2	H3	R17	R3	M17
A_{20}	Q8	D_{18}	C2	DP3	A5	S4	R6	P17
A_{21}	Q5	D_{19}	B1	EADS#	B17		R8	Q2
A_{22}	Q7	D_{20}	A1	FLUSH#	C15		R9	R4
A_{23}	S3	D_{21}	B2	HLDA	P15		R10	S6
A_{24}	Q6	D_{22}	A2	HOLD	E15		R11	S8
A_{25}	R2	D_{23}	A4	INTR	A16		R14	S9
A_{26}	S2	D_{24}	A6	KEN#	F15			S10
A_{27}	S1	D_{25}	B6	LOCK#	N15			S11
A_{28}	R1	D_{26}	C7	M/IO#	N16			S12
A_{29}	P2	D_{27}	C6	NMI	A15			S14
A_{30}	P3	D_{28}	C8	PCD	J17			
A_{31}	Q1	D_{29}	A8	PCHK#	Q17			
		D_{30}	C9	PWT	L15			
		D_{31}	B8	PLOCK#	Q16			
				RDY#	F16			
				RESET	C16			
				W/R#	N17			

NC pins should always remain unconnected, for other recommendations see Section 12.1.3.

Intel OverDrive™ PROCESSOR PIN CROSS REFERENCE

Table 1.1b. Intel OverDrive™ Processor Pin Cross Reference by Pin Name

Address		Data		Control		N/C	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	A3	B7	A7
A_3	R15	D_1	N2	ADS#	S17	A10	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	A12	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	A14	C4	B3
A_6	S15	D_4	M3	BE1#	J16	B10	C5	B4
A_7	Q13	D_5	J2	BE2#	J15	B12	E2	B5
A_8	R13	D_6	L2	BE3#	F17	B13	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16	B16	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17	C10	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15	C11	H16	G17
A_{12}	S7	D_{10}	E3	BREQ	Q15	C12	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16	C13	K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17	C14	K16	K1
A_{15}	R7	D_{13}	D2	CLK	C3	G15	L16	K17
A_{16}	Q9	D_{14}	K3	D/C#	M15	R17	M2	L1
A_{17}	Q3	D_{15}	F3	DP0	N3	S4	M16	L17
A_{18}	R5	D_{16}	J3	DP1	F1	D4	P16	M1
A_{19}	Q4	D_{17}	D3	DP2	H3		R3	M17
A_{20}	Q8	D_{18}	C2	DP3	A5		R6	P17
A_{21}	Q5	D_{19}	B1	EADS#	B17		R8	Q2
A_{22}	Q7	D_{20}	A1	FERR#	A13		R9	R4
A_{23}	S3	D_{21}	B2	FLUSH#	C15		R10	S6
A_{24}	Q6	D_{22}	A2	HLDA	P15		R11	S8
A_{25}	R2	D_{23}	A4	HOLD	E15		R14	S9
A_{26}	S2	D_{24}	A6	IGNNE#	A15			S10
A_{27}	S1	D_{25}	B6	INTR	A16			S11
A_{28}	R1	D_{26}	C7	KEN#	F15			S12
A_{29}	P2	D_{27}	C6	LOCK#	N15			S14
A_{30}	P3	D_{28}	C8	M/IO#	N16			
A_{31}	Q1	D_{29}	A8	UP#	B14			
		D_{30}	C9	NMI	B15			
		D_{31}	B8	PCD	J17			
				PCHK#	Q17			
				PWT	L15			
				PLOCK#	Q16			
				RDY#	F16			
				RESET	C16			
				W/R#	N17			

NC pins should always remain unconnected, for other recommendations see Section 12.1.3.

Table 1.1c. Complete Pin Reference of Intel486™ SX CPU (PQFP Package)

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	V_{SS}	50	V_{SS}	99	V_{SS}	148	V_{SS}
2	A_{21}	51	D_{21}	100	NMI	149	NC
3	A_{22}	52	NC	101	INTR	150	A_3
4	A_{23}	53	D_{22}	102	FLUSH#	151	NC
5	A_{24}	54	V_{CC}	103	RESET	152	A_4
6	V_{CC}	55	D_{23}	104	A20M#	153	NC
7	A_{25}	56	NC	105	EADS#	154	A_5
8	A_{26}	57	DP3	106	PCD	155	NC
9	A_{27}	58	V_{SS}	107	V_{CC}	156	UP#
10	A_{28}	59	D_{24}	108	PWT	157	NC
11	V_{SS}	60	NC	109	V_{SS}	158	A_6
12	A_{29}	61	D_{25}	110	D/C#	159	A_7
13	A_{30}	62	V_{CC}	111	M/IO#	160	NC
14	A_{31}	63	D_{26}	112	V_{CC}	161	A_8
15	NC	64	NC	113	BE3#	162	NC
16	DP0	65	D_{27}	114	V_{SS}	163	A_9
17	D_0	66	V_{SS}	115	BE2#	164	V_{CC}
18	D_1	67	D_{28}	116	BE1#	165	A_{10}
19	V_{CC}	68	NC	117	BE0#	166	NC
20	D_2	69	D_{29}	118	BREQ	167	V_{SS}
21	V_{SS}	70	V_{CC}	119	V_{CC}	168	V_{SS}
22	V_{SS}	71	D_{30}	120	W/R#	169	NC
23	D_3	72	NC	121	V_{SS}	170	V_{CC}
24	V_{CC}	73	NC	122	HLDA	171	NC
25	D_4	74	D_{31}	123	CLK	172	A_{11}
26	D_5	75	NC	124	NC	173	NC
27	D_6	76	NC	125	V_{CC}	174	A_{12}
28	V_{CC}	77	NC	126	V_{SS}	175	V_{CC}
29	D_7	78	NC	127	NC	176	A_{13}
30	DP1	79	NC	128	TCK	177	V_{SS}
31	D_8	80	TDO	129	AHOLD	178	A_{14}
32	D_9	81	NC	130	HOLD	179	V_{CC}
33	V_{SS}	82	NC	131	V_{CC}	180	A_{15}
34	NC	83	NC	132	KEN#	181	A_{16}
35	D_{10}	84	V_{CC}	133	RDY#	182	V_{SS}
36	V_{CC}	85	NC	134	NC	183	A_{17}
37	D_{11}	86	V_{SS}	135	BS8#	184	V_{CC}
38	D_{12}	87	NC	136	BS16#	185	TDI
39	D_{13}	88	NC	137	BOFF#	186	NC
40	V_{SS}	89	NC	138	BRDY#	187	TMS
41	D_{14}	90	NC	139	PCHK#	188	NC
42	D_{15}	91	NC	140	NC	189	A_{18}
43	DP2	92	NC	141	V_{SS}	190	NC
44	D_{16}	93	V_{CC}	142	LOCK#	191	A_{19}
45	D_{17}	94	NC	143	PLOCK#	192	NC
46	D_{18}	95	V_{SS}	144	BLAST#	193	A_{20}
47	D_{19}	96	V_{SS}	145	ADS#	194	V_{SS}
48	D_{20}	97	NC	146	A_2	195	NC
49	V_{CC}	98	V_{CC}	147	V_{CC}	196	V_{CC}

NC pins should always remain unconnected, for other recommendations see Section 12.1.3.

Table 1.1d. Pin Cross Reference by Signal Type (PQFP Package)

Address		Data		Control		NC	V_{CC}	V_{SS}
A_2	146	D_0	17	A20M#	104	15	6	1
A_3	150	D_1	18	ADS#	145	34	19	11
A_4	152	D_2	20	AHOLD	129	52	24	21
A_5	154	D_3	23	BE0#	117	56	28	22
A_6	158	D_4	25	BE1#	116	60	36	33
A_7	159	D_5	26	BE2#	115	64	49	40
A_8	161	D_6	27	BE3#	113	68	54	50
A_9	163	D_7	29	BLAST#	144	72	62	58
A_{10}	165	D_8	31	BOFF#	137	73	70	66
A_{11}	172	D_9	32	BRDY#	138	75	84	86
A_{12}	174	D_{10}	35	BREQ	118	76	93	95
A_{13}	176	D_{11}	37	BS8#	135	77	98	96
A_{14}	178	D_{12}	38	BS16#	136	78	107	99
A_{15}	180	D_{13}	39	CLK	123	79	112	109
A_{16}	181	D_{14}	41	D/C#	110	81	119	114
A_{17}	183	D_{15}	42	DP0	16	82	125	121
A_{18}	189	D_{16}	44	DP1	30	83	131	126
A_{19}	191	D_{17}	45	DP2	43	85	147	141
A_{20}	193	D_{18}	46	DP3	57	87	164	148
A_{21}	2	D_{19}	47	EADS#	105	88	170	167
A_{22}	3	D_{20}	48	FLUSH#	102	89	175	168
A_{23}	4	D_{21}	51	HLDA	122	90	179	177
A_{24}	5	D_{22}	53	HOLD	130	91	184	182
A_{25}	7	D_{23}	55	INTR	101	92	196	194
A_{26}	8	D_{24}	59	KEN#	132	94		
A_{27}	9	D_{25}	61	LOCK#	142	97		
A_{28}	10	D_{26}	63	M/IO#	111	124		
A_{29}	12	D_{27}	65	NMI	100	127		
A_{30}	13	D_{28}	67	PCD	106	134		
A_{31}	14	D_{29}	69	PCHK#	139	140		
		D_{30}	71	PWT	108	149		
		D_{31}	74	PLOCK#	143	151		
				RDY#	133	153		
				RESET	103	155		
				TDI	185	157		
				TDO	80	160		
				TMS	187	162		
				W/R#	120	166		
						169		
						171		
						173		
						186		
						188		
						190		
						192		
						195		

NC pins should always remain unconnected, for other recommendations see Section 12.1.3.

QUICK PIN REFERENCE

What follows is a brief pin description. For detailed signal descriptions refer to Section 6.

Symbol	Type	Name and Function
CLK	I	*Clock* provides the fundamental timing and the internal operating frequency for the Intel486 SX microprocessor/Intel OverDrive Processor. All external timing parameters are specified with respect to the rising edge of CLK.
ADDRESS BUS		
A31–A4 A2–A3	I/O O	A31–A2 are the *address lines* of the microprocessor. A31–A2, together with the byte enables BE0#–BE3#, define the physical area of memory or input/output space accessed. Address lines A31–A4 are used to drive addresses into the microprocessor to perform cache line invalidations. Input signals must meet setup and hold times t_{22} and t_{23}. A31–A2 are not driven during bus or address hold.
BE0–3#	O	The *byte enable* signals indicate active bytes during read and write cycles. During the first cycle of a cache fill, the external system should assume that all byte enables are active. BE3# applies to D24–D31, BE2# applies to D16–D23, BE1# applies to D8–D15 and BE0# applies to D0–D7. BE0#–BE3# are active LOW and are not driven during bus hold.
DATA BUS		
D31–D0	I/O	These are the *data lines* for the Intel486 SX microprocessor/Intel OverDrive Processor. Lines D0–D7 define the least significant byte of the data bus while lines D24–D31 define the most significant byte of the data bus. These signals must meet setup and hold times t_{22} and t_{23} for proper operation on reads. These pins are driven during the second and subsequent clocks of write cycles.
DATA PARITY		
DP0–DP3	I/O	There is one *data parity* pin for each byte of the data bus. Data parity is generated on all write data cycles with the same timing as the data driven by the Intel486 SX microprocessor/Intel OverDrive Processor. Even parity information must be driven back into the microprocessor on the data parity pins with the same timing as read information to insure that the correct parity check status is indicated by the Intel486 SX microprocessor/Intel OverDrive Processor. The signals read on these pins do not affect program execution. Input signals must meet setup and hold times t_{22} and t_{23}. DP0–DP3 should be connected to V_{CC} through a pullup resistor in systems which do not use parity. DP0–DP3 are active HIGH and are driven during the second and subsequent clocks of write cycles.
PCHK#	O	*Parity Status* is driven on the PCHK# pin the clock after ready for read operations. The parity status is for data sampled at the end of the previous clock. A parity error is indicated by PCHK# being LOW. Parity status is only checked for enabled bytes as indicated by the byte enable and bus size signals. PCHK# is valid only in the clock immediately after read data is returned to the microprocessor. At all other times PCHK# is inactive (HIGH). PCHK# is never floated.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
BUS CYCLE DEFINITION		
M/IO# D/C# W/R#	O O O	The *memory/input-output, data/control* and *write/read* lines are the primary bus definition signals. These signals are driven valid as the ADS# signal is asserted.
		M/IO# D/C# W/R# Bus Cycle Initiated 0 0 0 Interrupt Acknowledge 0 0 1 Halt/Special Cycle 0 1 0 I/O Read 0 1 1 I/O Write 1 0 0 Code Read 1 0 1 Reserved 1 1 0 Memory Read 1 1 1 Memory Write
		The bus definition signals are not driven during bus hold and follow the timing of the address bus. Refer to Section 7.2.11 for a description of the special bus cycles.
LOCK#	O	The *bus lock* pin indicates that the current bus cycle is locked. The Intel486 SX microprocessor/Intel OverDrive Processor will not allow a bus hold when LOCK# is asserted (but address holds are allowed). LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after the last clock of the last locked bus cycle. The last locked cycle ends when ready is returned. LOCK# is active LOW and is not driven during bus hold. Locked read cycles will not be transformed into cache fill cycles if KEN# is returned active.
PLOCK#	O	The *pseudo-lock* pin indicates that the current bus transaction requires more than one bus cycle to complete. For the Intel486 SX microprocessor/Intel OverDrive Processor, examples of such operations are segment table descriptor reads (64 bits), in addition to cache line fills (128 bits).
		For the Intel486 SX microprocessor/Intel OverDrive Processor, examples of such operations are floating point long reads and writes (64 bits), segment table descriptor reads (64 bits), in addition to cache line fills (128 bits).
		The Intel486 SX microprocessor/Intel OverDrive Processor will drive PLOCK# active until the addresses for the last bus cycle of the transaction have been driven regardless of whether RDY# or BRDY# have been returned.
		Normally PLOCK# and BLAST# are inverse of each other. However during the first bus cycle of a 64-bit floating point write, both PLOCK# and BLAST# will be asserted.
		PLOCK# is a function of the BS8#, BS16# and KEN# inputs. PLOCK# should be sampled only in the clock ready is returned. PLOCK# is active LOW and is not driven during bus hold.
BUS CONTROL		
ADS#	O	The *address status* output indicates that a valid bus cycle definition and address are available on the cycle definition lines and address bus. ADS# is driven active in the same clock as the addresses are driven. ADS# is active LOW and is not driven during bus hold.
RDY#	I	The *non-burst ready* input indicates that the current bus cycle is complete. RDY# indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted data from the Intel486 SX microprocessor/Intel OverDrive Processor in response to a write. RDY# is ignored when the bus is idle and at the end of the first clock of the bus cycle. RDY# is active during address hold. Data can be returned to the processor while AHOLD is active. RDY# is active LOW, and is not provided with an internal pullup resistor. RDY# must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
BURST CONTROL		
BRDY#	I	The *burst ready input* performs the same function during a burst cycle that RDY# performs during a non-burst cycle. BRDY# indicates that the external system has presented valid data in response to a read or that the external system has accepted data in response to a write. BRDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle. BRDY# is sampled in the second and subsequent clocks of a burst cycle. The data presented on the data bus will be strobed into the microprocessor when BRDY# is sampled active. If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is prematurely aborted. BRDY# is active LOW and is provided with a small pullup resistor. BRDY# must satisfy the setup and hold times t_{16} and t_{17}.
BLAST#	O	The *burst last* signal indicates that the next time BRDY# is returned the burst bus cycle is complete. BLAST# is active for both burst and non-burst bus cycles. BLAST# is active LOW and is not driven during bus hold.
INTERRUPTS		
RESET	I	The *reset* input forces the Intel486 SX microprocessor/Intel OverDrive Processor to begin execution at a known state. The microprocessor cannot begin execution of instructions until at least 1 ms after V_{CC} and CLK have reached their proper DC and AC specifications. The RESET pin should remain active during this time to insure proper microprocessor operation. RESET is active HIGH. RESET is asynchronous but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
INTR	I	The *maskable interrupt* indicates that an external interrupt has been generated. If the internal interrupt flag is set in EFLAGS, active interrupt processing will be initiated. The Intel486 SX microprocessor/Intel OverDrive Processor will generate two locked interrupt acknowledge bus cycles in response to the INTR pin going active. INTR must remain active until the interrupt acknowledges have been performed to assure that the interrupt is recognized. INTR is active HIGH and is not provided with an internal pulldown resistor. INTR is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
NMI	I	The *non-maskable interrupt* request signal indicates that an external non-maskable interrupt has been generated. NMI is rising edge sensitive. NMI must be held LOW for at least four CLK periods before this rising edge. NMI is not provided with an internal pulldown resistor. NMI is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
BUS ARBITRATION		
BREQ	O	The *bus request* signal indicates that the Intel486 SX microprocessor/Intel OverDrive Processor has internally generated a bus request. BREQ is generated whether or not the Intel486 SX microprocessor/Intel OverDrive Processor is driving the bus. BREQ is active HIGH and is never floated.
HOLD	I	The *bus hold request* allows another bus master complete control of the processor bus. In response to HOLD going active the Intel486 SX microprocessor/Intel OverDrive Processor will float most of its output and input/output pins. HLDA will be asserted after completing the current bus cycle, burst cycle or sequence of locked cycles. The Intel486 SX microprocessor/Intel OverDrive Processor will remain in this state until HOLD is deasserted. HOLD is active high and is not provided with an internal pulldown resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper operation.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
BUS ARBITRATION (Continued)		
HLDA	O	*Hold acknowledge* goes active in response to a hold request presented on the HOLD pin. HLDA indicates that the Intel486 SX microprocessor/Intel OverDrive Processor has given the bus to another local bus master. HLDA is driven active in the same clock that the Intel486 SX microprocessor/Intel OverDrive Processor floats its bus. HLDA is driven inactive when leaving bus hold. HLDA is active HIGH and remains driven during bus hold.
BOFF#	I	The *backoff* input forces the Intel486 SX microprocessor/Intel OverDrive Processor to float its bus in the next clock. The microprocessor will float all pins normally floated during bus hold but HLDA will not be asserted in response to BOFF#. BOFF# has higher priority than RDY# or BRDY#; if both are returned in the same clock, BOFF# takes effect. The microprocessor remains in bus hold until BOFF# is negated. If a bus cycle was in progress when BOFF# was asserted the cycle will be restarted. BOFF# is active LOW and must meet setup and hold times t_{18} and t_{19} for proper operation.
CACHE INVALIDATION		
AHOLD	I	The *address hold* request allows another bus master access to the processor's address bus for a cache invalidation cycle. The Intel486 SX microprocessor/Intel OverDrive Processor will stop driving its address bus in the clock following AHOLD going active. Only the address bus will be floated during address hold, the remainder of the bus will remain active. AHOLD is active HIGH and is provided with a small internal pulldown resistor. For proper operation AHOLD must meet setup and hold times t_{18} and t_{19}.
EADS#	I	This signal indicates that a *valid external address* has been driven onto the Intel486 SX microprocessor/Intel OverDrive Processor address pins. This address will be used to perform an internal cache invalidation cycle. EADS# is active LOW and is provided with an internal pullup resistor. EADS# must satisfy setup and hold times t_{12} and t_{13} for proper operation.
CACHE CONTROL		
KEN#	I	The *cache enable* pin is used to determine whether the current cycle is cacheable. When the Intel486 SX microprocessor/Intel OverDrive Processor generates a cycle that can be cached and KEN# is active one clock before RDY# or BRDY# during the first transfer of the cycle, the cycle will become a cache line fill cycle. Returning KEN# active one clock before RDY# during the last read in the cache line fill will cause the line to be placed in the on-chip cache. KEN# is active LOW and is provided with a small internal pullup resistor. KEN# must satisfy setup and hold times t_{14} and t_{15} for proper operation.
FLUSH#	I	The *cache flush* input forces the Intel486 SX microprocessor/Intel OverDrive Processor to flush its entire internal cache. FLUSH# is active low and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition in any specific clock. FLUSH# causes the Intel486 SX microprocessor/Intel OverDrive Processor to enter the tri-state test mode.
PAGE CACHEABILITY		
PWT PCD	O O	The *page write-through* and *page cache disable* pins reflect the state of the page attribute bits, PWT and PCD, in the page table entry, page directory entry or control register 3 (CR3) when paging is enabled. If paging is disabled, the CPU ignores the PCD and PWT bits and assumes they are zero for the purpose of caching and driving PCD and PWT pins. PWT and PCD have the same timing as the cycle definition pins (M/IO#, D/C#, and W/R#). PWT and PCD are active HIGH and are not driven during bus hold. PCD is masked by the cache disable bit (CD) in Control Register 0.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
NUMERIC ERROR REPORTING		
FERR#	O	The *floating point error* pin is driven active when a floating point error occurs. FERR# is similar to the ERROR# pin on the Intel387™ Math CoProcessor. FERR# is included for compatibility with systems using DOS type floating point error reporting. FERR# will not go active if FP errors are masked in FPU register. FERR# is active LOW, and is not floated during bus hold.
IGNNE#	I	When the *ignore numeric error* pin is asserted the Intel OverDrive Processor will ignore a numeric error and continue executing non-control floating point instructions, but FERR# will still be activated by the Intel OverDrive Processor When IGNNE# is deasserted the Intel OverDrive Processor will freeze on a non-control floating point instruction, if a previous floating point instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set. IGNNE# is active LOW and is provided with a small internal pullup resistor. IGNNE# is asynchronous but setup and hold times t_{20} and t_{21} must be met to insure recognition on any specific clock.
INTEL OverDrive™ PROCESSOR INTERFACE		
UP#	O	The upgrade present pin is used to signal the Intel486 microprocessor to float its outputs and get-off the bus. This pin can also be used to check the presence of the OverDrive Processor in the two socket upgrade circuit. It is active low and is never floated. UP# is driven low at power-up and remain active for the entire duration of the Intel OverDrive Processor operation.
KEY PIN		
KEY		The KEY pin is an electrically non-functional pin which is used to insure correct Intel OverDrive Processor orientation in a 169 pin socket.
BUS SIZE CONTROL		
BS16# BS8#	I I	The *bus size 16* and *bus size 8* pins (bus sizing pins) cause the Intel486 SX microprocessor/Intel OverDrive Processor to run multiple bus cycles to complete a request from devices that cannot provide or accept 32 bits of data in a single cycle. The bus sizing pins are sampled every clock. The state of these pins in the clock before ready is used by the Intel486 SX microprocessor/Intel OverDrive Processor to determine the bus size. These signals are active LOW and are provided with internal pullup resistors. These inputs must satisfy setup and hold times t_{14} and t_{15} for proper operation.
ADDRESS MASK		
A20M#	I	When the address bit 20 mask pin is asserted, the Intel486 SX microprocessor/Intel OverDrive Processer masks physical address bit 20 (A20) before performing a lookup to the internal cache or driving a memory cycle on the bus. A20M# emulates the address wraparound at one Mbyte which occurs on the 8086. A20M# is active LOW and should be asserted only when the processor is in real mode. This pin is asynchrous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For proper operation, A20M# should be sampled high at the falling edge of RESET.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
PERFORMANCE UPGRADE SUPPORT (Intel486 SX CPU PQFP Version Only)		
UP#	I	*Upgrade Present* forces the Intel486 SX CPU to tri-state all its outputs and enter the power down mode. UP# is active low and is sampled at all times, including after power-up and during reset. UP# is provided with an internal pull-up resistor.
TEST ACCESS PORT (Intel486 SX CPU PQFP Version Only)		
TCK	I	*Test Clock* is an input to the Intel486™ CPU and provides the clocking function required by the JTAG Boundary scan feature. TCK is used to clock state information and data into component on the rising edge of TCK on TMS and TDI, respectively. Data is clocked out of the part on the falling edge of TCK and TDO. TCK is provided with an internal pull-up resistor.
TDI	I	*Test Data Input* is the serial input used to shift JTAG instructions and data into component. TDI is sampled on the rising edge of TCK, during the SHIFT-IR and SHIFT-DR TAP controller states. During all other tap controller states, TDI is a "don't care". TDI is provided with an internal pull-up resistor.
TDO	O	*Test Data Output* is the serial output used to shift JTAG instructions and data out of the component. TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT-DR TAP controller states. At all other times TDO is driven to the high impedance state.
TMS	I	*Test Mode Select* is decoded by the JTAG TAP (Tap Access Port) to select the operation of the test logic. TMS is sampled on the rising edge of TCK. To guarantee deterministic behavior of the TAP controller TMS is provided with an internal pull-up resistor.

Table 1.2. Output Pins

Name	Active Level	When Floated
BREQ	HIGH	
HLDA	HIGH	
BE0#–BE3#	LOW	Bus Hold
PWT, PCD	HIGH	Bus Hold
W/R#, D/C#, M/IO#	HIGH	Bus Hold
LOCK#	LOW	Bus Hold
PLOCK#	LOW	Bus Hold
ADS#	LOW	Bus Hold
BLAST#	LOW	Bus Hold
PCHK#	LOW	
FERR#	LOW	
A2–A3	HIGH	Bus, Address Hold
MP#	LOW	

Table 1.3. Input Pins

Name	Active Level	Synchronous/ Asynchronous
CLK		
RESET	HIGH	Asynchronous
HOLD	HIGH	Synchronous
AHOLD	HIGH	Synchronous
EADS#	LOW	Synchronous
BOFF#	LOW	Synchronous
FLUSH#	LOW	Asynchronous
A20M#	LOW	Asynchronous
BS16#, BS8#	LOW	Synchronous
KEN#	LOW	Synchronous
RDY#	LOW	Synchronous
BRDY#	LOW	Synchronous
INTR	HIGH	Asynchronous
NMI	HIGH	Asynchronous
IGNNE#	LOW	Asynchronous
UP#(1)	LOW	Asynchronous

NOTE:
1. The UP# pin is present on the Intel486 SX CPU PQFP package only.

Table 1.4. Input/Output Pins

Name	Active Level	When Floated
D0–D31	HIGH	Bus Hold
DP0–DP3	HIGH	Bus Hold
A4–A31	HIGH	Bus, Address Hold

Table 1.5. Test Pins

Name	Input or Output	Sampled/ Driven On
TCK	Input	N/A
TDI	Input	Rising Edge of TCK
TDO	Output	Falling Edge of TCK
TMS	Input	Rising Edge of TCK

Table 1.6. Component and Revision ID

Product	Stepping	Component ID	Revision ID
i486 SX Microprocessor	A0	04	20
	B0	04	22
	cA0	04	27
	cB0	04	28
i487 SX Math CoProcessor	A0	04	20
	B0	04	21
OverDrive Processor	A2	04	32
	B1	04	33

NOTE:
Table 1.6 shows the Component ID number and Revision ID number for the B-0 stepping of the Intel486 SX Microprocessor and Intel OverDrive Processor. When an Intel OverDrive Processor is installed in the system, the Component ID and Revision ID is provided by the Intel OverDrive Processor and not the Intel486 SX Microprocessor. The Component ID and Revision ID read by the BIOS/software may change when an OverDrive Processor is installed in an Intel486 SX Microprocessor based system.

2.0 ARCHITECTURAL OVERVIEW

The Intel486 SX microprocessor/Intel OverDrive Processor is a 32-bit architecture with on-chip memory management, and cache memory units.

The Intel OverDrive Processor is the upgrade for the Intel486 SX microprocessor. It is a 32-bit architecture with on-chip memory management, floating point and cache memory units.

The Intel486 SX microprocessor/Intel OverDrive Processor contains all the features of the Intel386 microprocessor with enhancements to increase performance. The instruction set includes the complete Intel386 microprocessor instruction set along with extensions to serve new applications. The on-chip memory management unit (MMU) is completely compatible with the Intel386 microprocessor MMU. All software written for the Intel386 microprocessor and previous members of the Intel386™/Intel486™ architectural family will run on the Intel486 SX microprocessor/Intel OverDrive Processor without any modifications.

The Intel OverDrive Processor brings the 387 Math CoProcessor on-chip. All software written for the Intel386 microprocessor, Intel387 Math CoProcessor or Intel486 DX microprocessor and previous members of the 86/87 architectural family will run on the Intel OverDrive Processor without any modifications.

Several enhancements have been added to increase performance. On-chip cache memory allows frequently used data and code to be stored on-chip reducing accesses to the external bus. RISC design techniques have been used to reduce instruction cycle times. A burst bus feature enables fast cache fills. All of these features, combined, lead to performance greater than twice that of an Intel386 microprocessor.

The memory management unit (MMU) consists of a segmentation unit and a paging unit. Segmentation allows management of the logical address space by providing easy data and code relocatibility and efficient sharing of global resources. The paging mechanism operates beneath segmentation and is transparent to the segmentation process. Paging is optional and can be disabled by system software. Each segment can be divided into one or more 4 Kbyte segments. To implement a virtual memory system, full restartability for all page and segment faults is supported.

Memory is organized into one or more variable length segments, each up to four gigabytes (2^{32} bytes) in size. A segment can have attributes associated with it which include its location, size, type (i.e., stack, code or data), and protection characteristics. Each task on an Intel486 SX microprocessor/Intel OverDrive Processor can have a maximum of 16,381 segments, each up to four gigabytes in size. Thus each task has a maximum of 64 terabytes (trillion bytes) of virtual memory.

The segmentation unit provides four-levels of protection for isolating and protecting applications and the operating system from each other. The hardware enforced protection allows the design of systems with a high degree of integrity.

The Intel486 SX microprocessor/Intel OverDrive Processor has two modes of operation: Real Address Mode (Real Mode) and Protected Mode Virtual Address Mode (Protected Mode). In Real Mode the Intel486 SX microprocessor/Intel OverDrive Processor operates as a very fast 8086. Real Mode is required primarily to set up the Intel486 SX microprocessor/Intel OverDrive Processor for Protected Mode operation. Protected Mode provides access to the sophisticated memory management paging and privilege capabilities of the processor.

Within Protected Mode, software can perform a task switch to enter into tasks designated as Virtual 8086 Mode tasks. Each virtual 8086 task behaves with 8086 semantics, allowing 8086 software (an application program or an entire operating system) to execute.

The on-chip floating point unit operates in parallel with the arithmetic and logic unit and provides arithmetic instructions for a variety of numeric data types. It executes numerous built-in transcendental functions (e.g., tangent, sine, cosine, and log functions). The floating point unit fully conforms to the ANSI/IEEE standard 754-1985 for floating point arithmetic.

The on-chip cache is 8 Kbytes in size. It is 4-way set associative and follows a write-through policy. The on-chip cache includes features to provide flexibility in external memory system design. Individual pages can be designated as cacheable or non-cacheable by software or hardware. The cache can also be enabled and disabled by software or hardware.

Finally the Intel486 SX microprocessor/Intel OverDrive Processor have features to facilitate high performance hardware designs. The 1X clock input eases high frequency board level designs. The burst bus feature enables fast cache fills. These features are described beginning in Section 6.

2.0.1 PQFP PACKAGE

The Intel486 SX microprocessor is also available in PQFP package. The PQFP version is compatible to the one in PGA. Additionally, it offers the following new and/or improved features:

- Boundary Scan (JTAG)
- Reduced Power Dissipation
- Power Down Mode
- Lower Input Capacitance
- New Pin to Support the performance upgrade components

Boundary Scan (JTAG)

Boundary Scan is discussed in detail in Section 8.5.

Lower Power Supply Current (I_{CC})

See DC specifications.

New Pin to Support the OverDrive Processor

The PQFP version of the Intel486 SX CPU provides a direct interface with the OverDrive Processor. Thus, the interface logic between the Intel486 SX CPU and the Intel487 SX MCP, as shown in Figure 6.6, is not required. Instead, the UP# pin of the Intel487 MCP is connected directly to the UP# pin of the Intel486 SX CPU. All other signals are connected together as shown in Figure 6.7.

Power Down Mode

Power Down Mode is a new mode in the Intel486 SX CPU in PQFP. It is initiated by the Intel OverDrive Processor, via the Upgrade Circuit. Upon sensing the presence of the Intel OverDrive Processor, the Intel486 SX microprocessor not only tri-states its output but also enters the "Power Down Mode", whereby it remains tri-stated and lowers its power consumption (see D.C. specifications).

The Intel486 SX CPU samples the UP# pin to enter the Power Down Mode (PDM). The UP# pin of the Intel486 SX microprocessor is driven active by the UP# pin of Intel OverDrive Processor in the Performance Upgrade Circuit, as shown in Figure 6.7. More details on power down mode are given in Section 6.6.2.1.

Lower Input Capacitance

See DC specifications.

2.1 Register Set

The Intel486 SX microprocessor register set includes all the registers contained in the Intel386 microprocessor.

The Intel OverDrive Processor register set includes all the registers contained in the Intel386 microprocessor and the Intel387 Math CoProcessor.

The register set can be split into the following categories:

- Base Architecture Registers
  - General Purpose Registers
  - Instruction Pointer
  - Flags Register
  - Segment Registers
- Systems Level Registers
  - Control Registers
  - System Address Registers
- Floating Point Registers
  - Data Registers
  - Tag Word
  - Status Word
  - Instruction and Data Pointers
  - Control Word
- Debug and Test Registers

The base architecture and floating point registers are accessible by the applications program.

The floating point registers are accessible in the Intel OverDrive Processor.

The system level registers are only accessible at privilege level 0 and are used by the systems level program. The debug and test registers are also only accessible at privilege level 0.

2.1.1 BASE ARCHITECTURE REGISTERS

Figure 2.1 shows the Intel486 SX microprocessor/ Intel OverDrive Processor base architecture registers. The contents of these registers are task-specific and are automatically loaded with a new context upon a task switch operation.

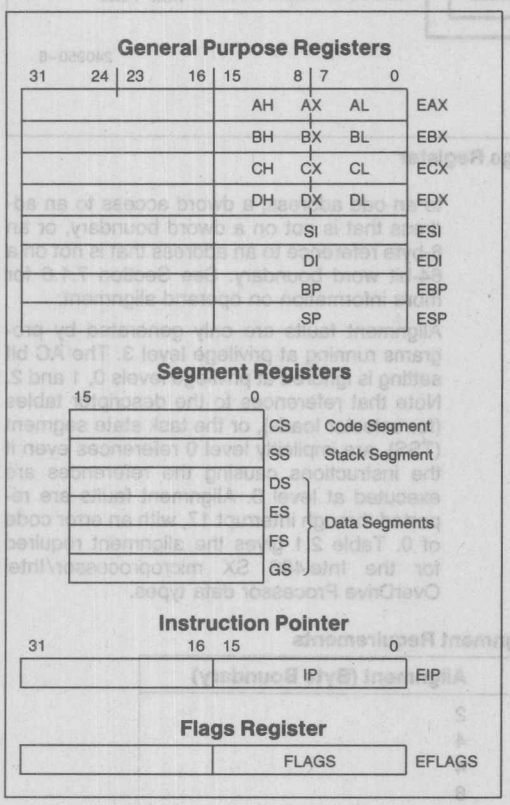


Figure 2.1. Base Architecture Registers

The base architecture includes six directly accessible descriptors, each specifying a segment up to 4 Gbytes in size. The descriptors are indicated by the selector values placed in the Intel486 SX microprocessor/Intel OverDrive Processor segment registers. Various selector values can be loaded as a program executes.

The selectors are also task-specific, so the segment registers are automatically loaded with new context upon a task switch operation.

2.1.1.1 General Purpose Registers

The eight 32-bit general purpose registers are shown in Figure 2.1. These registers hold data or address quantities. The general purpose registers can support data operands of 1, 8, 16 and 32 bits, and bit fields of 1 to 32 bits. Address operands of 16 and 32 bits are supported. The 32-bit registers are named EAX, EBX, ECX, EDX, ESI, EDI, EBP and ESP.

The least significant 16 bits of the general purpose registers can be accessed separately by using the 16-bit names of the registers AX, BX, CX, DX, SI, DI, BP and SP. The upper 16 bits of the register are not changed when the lower 16 bits are accessed separately.

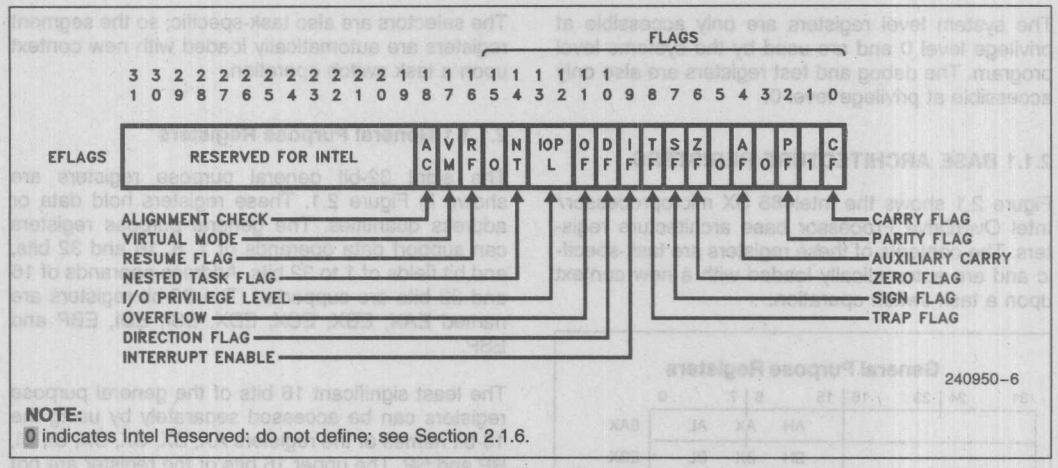
Finally 8-bit operations can individually access the lowest byte (bits 0–7) and the higher byte (bits 8–15) of the general purpose registers AX, BX, CX and DX. The lowest bytes are named AL, BL, CL and DL respectively. The higher bytes are named AH, BH, CH and DH respectively. The individual byte accessibility offers additional flexibility for data operations but is not used for effective address calculation.

2.1.1.2 Instruction Pointer

The instruction pointer, shown in Figure 2.1, is a 32-bit register named EIP. EIP holds the offset of the next instruction to be executed. The offset is always relative to the base of the code segment (CS). The lower 16 bits (bits 0–15) of the EIP contain the 16-bit instruction pointer named IP, which is used for 16-bit addressing.

2.1.1.3 Flags Register

The flags register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS control certain operations and indicate status of the Intel486 SX microprocessor/Intel OverDrive Processor. The lower 16 bits (bit 0–15) of EFLAGS contain the 16-bit register named FLAGS, which is most useful when executing 8086 and 80286 code. EFLAGS is shown in Figure 2.2.



NOTE:
0 indicates Intel Reserved: do not define; see Section 2.1.6.

Figure 2.2. Flags Register

EFLAGS bits 1, 3, 5, 15 and 19–31 are "undefined". When these bits are stored during interrupt processing or with a PUSHF instruction (push flags onto stack), a one is stored in bit 1 and zeros in bits 3, 5, 15 and 19–31.

The EFLAGS register in the Intel486 SX microprocessor/Intel OverDrive Processor contain a new bit not previously defined. The new bit, AC, is defined in the upper 16 bits of the register and it enables faults on accesses to misaligned data.

AC (Alignment Check, bit 18)

The AC bit enables the generation of faults if a memory reference is to a misaligned address. Alignment faults are enabled when AC is set to 1. A misaligned address is a word access to an odd address, a dword access to an address that is not on a dword boundary, or an 8-byte reference to an address that is not on a 64-bit word boundary. See Section 7.1.6 for more information on operand alignment.

Alignment faults are only generated by programs running at privilege level 3. The AC bit setting is ignored at privilege levels 0, 1 and 2. Note that references to the descriptor tables (for selector loads), or the task state segment (TSS), are implicitly level 0 references even if the instructions causing the references are executed at level 3. Alignment faults are reported through interrupt 17, with an error code of 0. Table 2.1 gives the alignment required for the Intel486 SX microprocessor/Intel OverDrive Processor data types.

Table 2.1. Data Type Alignment Requirements

Memory Access	Alignment (Byte Boundary)
Word	2
Dword	4
Single Precision Real	4
Double Precision Real	8
Extended Precision Real	8
Selector	2
48-Bit Segmented Pointer	4
32-Bit Flat Pointer	4
32-Bit Segmented Pointer	2
48-Bit "Pseudo-Descriptor"	4
FSTENV/FLDENV Save Area	4/2 (On Operand Size)
FSAVE/FRSTOR Save Area	4/2 (On Operand Size)
Bit String	4

IMPLEMENTATION NOTE:
Several instructions on the Intel486 SX microprocessor/Intel OverDrive Processor generate misaligned references, even if their memory address is aligned. For example, on the Intel486 SX microprocessor/Intel OverDrive Processor, the SGDT/SIDT (store global/interrupt descriptor table) instruction reads/writes two bytes, and then reads/writes four bytes from a "pseudo-descriptor" at the given address. The Intel486 SX microprocessor/Intel OverDrive Processor will generate misaligned references unless the address is on a 2 mod 4 boundary. The FSAVE and FRSTOR instructions (floating point save and restore state) will generate misaligned references for one-half of the register save/restore cycles. The Intel486 SX microprocessor/Intel OverDrive Processor will not cause any AC faults if the effective address given in the instruction has the proper alignment.

VM (Virtual 8086 Mode, bit 17)

The VM bit provides Virtual 8086 Mode within Protected Mode. If set while the Intel486 SX microprocessor/Intel OverDrive Processor is in Protected Mode, the Intel486 SX microprocessor/Intel OverDrive Processor will switch to Virtual 8086 operation, handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes. The VM bit can be set only in Protected Mode, by the IRET instruction (if current privilege level = 0) and by task switches at any privilege level. The VM bit is unaffected by POPF. PUSHF always pushes a 0 in this bit, even if executing in Virtual 8086 Mode. The EFLAGS image pushed during interrupt processing or saved during task switches will contain a 1 in this bit if the interrupted code was executing as a Virtual 8086 Task.

RF (Resume Flag, bit 16)

The RF flag is used in conjunction with the debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. When RF is set, it causes any debug fault to be ignored on the next instruction. RF is then automatically reset at the successful completion of every instruction (no faults are signalled) except the IRET instruction, the POPF instruction, (and JMP, CALL, and INT instructions causing a task switch). These instructions set RF to the value specified by the memory image. For example, at the end of the breakpoint service routine, the IRET instruction can pop an EFLAG image having the RF bit set and resume the program's execution at the breakpoint address without generating another breakpoint fault on the same location.

NT (Nested Task, bit 14)

This flag applies to Protected Mode. NT is set to indicate that the execution of this task is nested within another task. If set, it indicates that the current nested task's Task State Segment (TSS) has a valid back link to the previous task's TSS. This bit is set or reset by control transfers to other tasks. The value of NT in EFLAGS is tested by the IRET instruction to determine whether to do an inter-task return or an intra-task return. A POPF or an IRET instruction **will** affect the setting of this bit according to the image popped, at any privilege level.

IOPL (Input/Output Privilege Level, bits 12-13)

This two-bit field applies to Protected Mode. IOPL indicates the numerically maximum CPL (current privilege level) value permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O Permission Bitmap. It also indicates the maximum CPL value allowing alteration of the IF (INTR Enable Flag) bit when new values are popped into the EFLAG register. POPF and IRET instruction can alter the IOPL field when executed at CPL = 0. Task switches can always alter the IOPL field, when the new flag image is loaded from the incoming task's TSS.

OF (Overflow Flag, bit 11)

OF is set if the operation resulted in a signed overflow. Signed overflow occurs when the operation resulted in carry/borrow **into** the sign bit (high-order bit) of the result but did not result in a carry/borrow **out of** the high-order bit, or vice-versa. For 8-, 16-, 32-bit operations, OF is set according to overflow at bit 7, 15, 31, respectively.

DF (Direction Flag, bit 10)

DF defines whether ESI and/or EDI registers postdecrement or postincrement during the string instructions. Postincrement occurs if DF is reset. Postdecrement occurs if DF is set.

IF (INTR Enable Flag, bit 9)

The IF flag, when set, allows recognition of external interrupts signalled on the INTR pin. When IF is reset, external interrupts signalled on the INTR are not recognized. IOPL indicates the maximum CPL value allowing alteration of the IF bit when new values are popped into EFLAGS or FLAGS.

TF (Trap Enable Flag, bit 8)

TF controls the generation of exception 1 trap when single-stepping through code. When TF is set, the Intel486 SX microprocessor/Intel OverDrive Processor generates an exception 1 trap after the next instruction is executed. When TF is reset, exception 1 traps occur only as a function of the breakpoint addresses loaded into debug registers DR0–DR3.

SF (Sign Flag, bit 7)

SF is set if the high-order bit of the result is set, it is reset otherwise. For 8-, 16-, 32-bit operations, SF reflects the state of bit 7, 15, 31 respectively.

ZF (Zero Flag, bit 6)

ZF is set if all bits of the result are 0. Otherwise it is reset.

AF (Auxiliary Carry Flag, bit 4)

The Auxiliary Flag is used to simplify the addition and subtraction of packed BCD quantities. AF is set if the operation resulted in a carry out of bit 3 (addition) or a borrow into bit 3 (subtraction). Otherwise AF is reset. AF is affected by carry out of, or borrow into bit 3 only, regardless of overall operand length: 8, 16 or 32 bits.

PF (Parity Flags, bit 2)

PF is set if the low-order eight bits of the operation contains an even number of "1's" (even parity). PF is reset if the low-order eight bits have odd parity. PF is a function of only the low-order eight bits, regardless of operand size.

CF (Carry Flag, bit 0)

CF is set if the operation resulted in a carry out of (addition), or a borrow into (subtraction) the high-order bit. Otherwise CF is reset. For 8-, 16- or 32-bit operations, CF is set according to carry/borrow at bit 7, 15 or 31, respectively.

NOTE:
In these descriptions, "set" means "set to 1," and "reset" means "reset to 0."

2.1.1.4 Segment Registers

Six 16-bit segment registers hold segment selector values identifying the currently addressable memory segments. In protected mode, each segment may range in size from one byte up to the entire linear and physical address space of the machine, 4 Gbytes (2^{32} bytes). In real address mode, the maximum segment size is fixed at 64 Kbytes (2^{16} bytes).

The six addressable segments are defined by the segment registers CS, SS, DS, ES, FS and GS. The selector in CS indicates the current code segment; the selector in SS indicates the current stack segment; the selectors in DS, ES, FS and GS indicate the current data segments.

2.1.1.5 Segment Descriptor Cache Registers

The segment descriptor cache registers are not programmer visible, yet it is very useful to understand their content. A programmer invisible descriptor cache register is associated with each programmer-visible segment register, as shown by Figure 2.3. Each descriptor cache register holds a 32-bit base address, a 32-bit segment limit, and the other necessary segment attributes.

SEGMENT REGISTERS		DESCRIPTOR REGISTERS (LOADED AUTOMATICALLY)														
15 0		Physical Base Address						Segment Limit					Other Segment Attributes from Descriptor			
Selector	CS–													—		
Selector	SS–													—		—
Selector	DS–													—	—	—
Selector	ES–													—	—	—
Selector	FS–													—	—	—
Selector	GS–													—	—	—

Figure 2.3. Intel486™ SX Microprocessor/Intel OverDrive Processor Segment Registers and Associated Descriptor Cache Registers

When a selector value is loaded into a segment register, the associated descriptor cache register is automatically updated with the correct information. In Real Address Mode, only the base address is updated directly (by shifting the selector value four bits to the left), since the segment maximum limit and attributes are fixed in Real Mode. In Protected Mode, the base address, the limit, and the attributes are all updated per the contents of the segment descriptor indexed by the selector.

Whenever a memory reference occurs, the segment descriptor cache register associated with the segment being used is automatically involved with the memory reference. The 32-bit segment base address becomes a component of the linear address calculation, the 32-bit limit is used for the limit-check operation, and the attributes are checked against the type of memory reference requested.

2.1.2 SYSTEM LEVEL REGISTERS

The system level registers, Figure 2.4, control operation of the on-chip cache, the on-chip floating point unit (FPU) and the segmentation and paging mechanisms.

Only the Intel OverDrive Processor contains the on-chip floating point unit (FPU).

The Intel486 SX microprocessor contains all the system level registers except the floating point unit. These registers are only accessible to programs running at privilege level 0, the highest privilege level.

The system level registers include three control registers and four segmentation base registers. The three control registers are CR0, CR2 and CR3. CR1 is reserved for future Intel processors. The four segmentation base registers are the Global Descriptor Table Register (GDTR), the Interrupt Descriptor Table Register (IDTR), the Local Descriptor Table Register (LDTR) and the Task State Segment Register (TR).

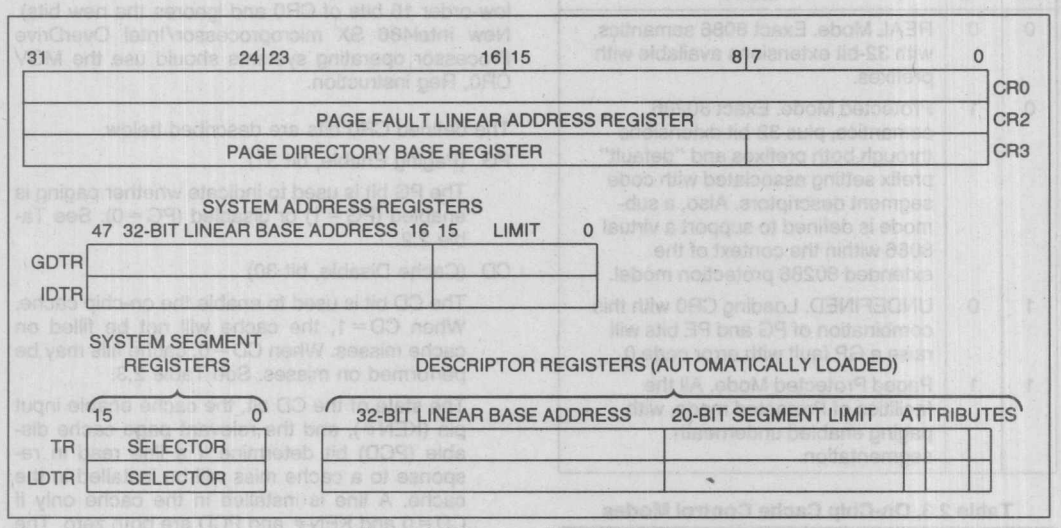


Figure 2.4. System Level Registers

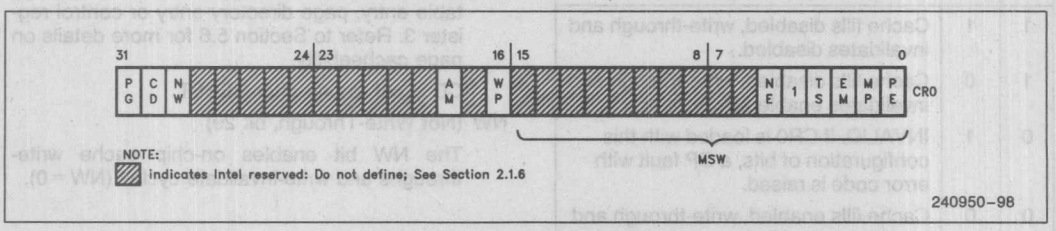


Figure 2.5. Control Register 0

2.1.2.1 Control Registers

Control Register 0 (CR0)

CR0, shown in Figure 2.5, contains 10 bits for control and status purposes. Five of the bits defined in the Intel486 SX microprocessor/Intel OverDrive Processor CR0 are newly defined. The new bits are CD, NW, AM, WP and NE. The function of the bits in CR0 can be categorized as follows:

Intel486 SX Microprocessor/Intel OverDrive Processor Operating Modes: PG, PE (Table 2.2)

On-Chip Cache Control Modes: CD, NW (Table 2.3)

On-Floating Point Unit Control: TS, EM, MP, NE (Table 2.4)

Alignment Check Control: AM

Supervisor Write Protect: WP

Table 2.2. Intel486™ SX Microprocessor/ Intel OverDrive™ Processor Operating Modes

PG	PE	Mode
0	0	REAL Mode. Exact 8086 semantics, with 32-bit extensions available with prefixes.
0	1	Protected Mode. Exact 80286 semantics, plus 32-bit extensions through both prefixes and "default" prefix setting associated with code segment descriptors. Also, a sub-mode is defined to support a virtual 8086 within the context of the extended 80286 protection model.
1	0	UNDEFINED. Loading CR0 with this combination of PG and PE bits will raise a GP fault with error code 0.
1	1	Paged Protected Mode. All the facilities of Protected mode, with paging enabled underneath segmentation.

Table 2.3. On-Chip Cache Control Modes

CD	NW	Operating Mode
1	1	Cache fills disabled, write-through and invalidates disabled.
1	0	Cache fills disabled, write-through and invalidates enabled.
0	1	INVALID. If CR0 is loaded with this configuration of bits, a GP fault with error code is raised.
0	0	Cache fills enabled, write-through and invalidates enabled.

Table 2.4. Intel OverDrive™ Processor Floating Point Instruction Control

CR0 BIT			Instruction Type	
EM	TS	MP	Floating-Point	Wait
0	0	0	Execute	Execute
0	0	1	Execute	Execute
0	1	0	Trap 7	Execute
0	1	1	Trap 7	Trap 7
1	0	0	Trap 7	Execute
1	0	1	Trap 7	Execute
1	1	0	Trap 7	Execute
1	1	1	Trap 7	Trap 7

The low-order 16 bits of CR0 are also known as the Machine Status Word (MSW), for compatibility with the 80286 protected mode. LMSW and SMSW (load and store MSW) instructions are taken as special aliases of the load and store CR0 operations, where only the low-order 16 bits of CR0 are involved. The LMSW and SMSW instructions in the Intel486 SX microprocessor/Intel OverDrive Processor work in an identical fashion to the LMSW and SMSW instructions in the 80286 (i.e., they only operate on the low-order 16 bits of CR0 and ignores the new bits). New Intel486 SX microprocessor/Intel OverDrive Processor operating systems should use the MOV CR0, Reg instruction.

The defined CR0 bits are described below.

PG (Paging Enable, bit 31)

The PG bit is used to indicate whether paging is enabled (PG = 1) or disabled (PG = 0). See Table 2.2.

CD (Cache Disable, bit 30)

The CD bit is used to enable the on-chip cache. When CD = 1, the cache will not be filled on cache misses. When CD = 0, cache fills may be performed on misses. See Table 2.3.

The state of the CD bit, the cache enable input pin (KEN#), and the relevant page cache disable (PCD) bit determine if a line read in response to a cache miss will be installed in the cache. A line is installed in the cache only if CD = 0 and KEN# and PCD are both zero. The relevant PCD bit comes from either the page table entry, page directory entry or control register 3. Refer to Section 5.6 for more details on page cacheability.

CD is set to one after RESET.

NW (Not Write-Through, bit 29)

The NW bit enables on-chip cache write-throughs and write-invalidate cycles (NW = 0).

When NW = 0, all writes, including cache hits, are sent out to the pins. Invalidate cycles are enabled when NW = 0. During an invalidate cycle a line will be removed from the cache if the invalidate address hits in the cache. See Table 2.3.

When NW = 1, write-throughs and write-invalidate cycles are disabled. A write will not be sent to the pins if the write hits in the cache. With NW = 1 the only write cycles that reach the external bus are cache misses. Write hits with NW = 1 will never update main memory. Invalidate cycles are ignored when NW = 1.

AM (Alignment Mask, bit 18)

The AM bit controls whether the alignment check (AC) bit in the flag register (EFLAGS) can allow an alignment fault. AM = 0 disables the AC bit. AM = 1 enables the AC bit. AM = 0 is the Intel386 microprocessor compatible mode.

Intel386 microprocessor software may load incorrect data into the AC bit in the EFLAGS register. Setting AM = 0 will prevent AC faults from occurring before the Intel486 SX microprocessor/Intel OverDrive Processor has created the AC interrupt service routine.

WP (Write Protect, bit 16)

WP protects read-only pages from supervisor write access. The Intel386 microprocessor allows a read-only page to be written from privilege levels 0–2. The Intel486 SX microprocessor/Intel OverDrive Processor are compatible with the Intel386 microprocessor when WP = 0. WP = 1 forces a fault on a write to a read-only page from any privilege level. Operating systems with Copy-on-Write features can be supported with the WP bit. Refer to Section 4.5.3 for further details on use of the WP bit.

NE (Numerics Exception, bit 5)

For the Intel486 SX microprocessor, interrupt 7 will be generated upon encountering any floating point instruction regardless of the value of the NE bit. It is recommended that NE = 1 for normal operation of the Intel486 SX microprocessor.

For the Intel OverDrive Processor, the NE bit controls whether unmasked floating point exceptions (UFPE) are handled through interrupt vector 16 (NE = 1) or through an external interrupt (NE = 0). NE = 0 (default at reset) supports the DOS operating system error reporting scheme from the 8087, 80287 and Intel387 Math CoProcessor. In DOS systems, math coprocessor errors are reported via external interrupt vector 13. DOS uses interrupt vector 16 for an operating system call. Refer to Sections 6.2.13 and 7.2.14 for more information on floating point error reporting.

For any UFPE the floating point error output pin (FERR#) will be driven active.

For NE = 0, the Intel OverDrive Processor works in conjunction with the ignore numeric error input (IGNNE#) and the FERR# output pins. When a UFPE occurs and the IGNNE# input is inactive, the Intel OverDrive Processor freezes immediately before executing the next floating point instruction. An external interrupt controller will supply an interrupt vector when FERR# is driven active. The UFPE is ignored if IGNNE# is active and floating point execution continues.

NOTE:
The freeze does not take place if the next instruction is one of the control instructions FNCLEX, FNINIT, FNSAVE, FNSTENV, FNSTCW, FNSTSW, FNSTSW AX, FNENI, FNDISI and FNSETPM. The freeze does occur if the next instruction is WAIT.

For NE = 1, any UFPE will result in a software interrupt 16, immediately before executing the next non-control floating point or WAIT instruction. The ignore numeric error input (IGNNE#) signal will be ignored.

PE (Protection Enable, bit 0)

The PE bit enables the segment based protection mechanism if PE = 1 protection is enabled. When PE = 0 the Intel486 SX microprocessor/ Intel OverDrive Processor operates in REAL mode, with segment based protection disabled, and addresses formed as in an 8086. Refer to Table 2.2.

For the Intel486 SX microprocessor, the following descriptions for EM, MP and TS apply.

EM (Emulate Coprocessor, bit 2)

EM bit should be set to one for the Intel486 SX microprocessor. This will cause the Intel486 SX microprocessor to trap via interrupt vector 7 (Device Not Available) to a software exception handler whenever it encounters a floating point instruction. If EM bit is 0, the system will hang.

MP (Monitor Coprocessor, bit 1)

For normal operation of the Intel486 SX microprocessor it is required to set this bit as zero (MP = 0). The MP bit is used in conjunction with the TS bit to determine if WAIT instructions should trap. For MP = 0, the value of TS is a don't care for these type of instructions (see Table 2.5a).

TS (Task Switch, bit 3)

The TS bit is set whenever a task switch operation is performed. Execution of floating point instructions with TS = 1 will cause a Device Not Available (DNA) fault (trap vector 7). With MP = 0, the value of TS bit is a don't care for the WAIT instructions i.e., these instructions will not generate trap 7 (see Table 2.5a).

Table 2.5a. Recommended Values of EM and MP Bits in CR0 Register for Intel486™ SX Microprocessor

CR0 Bit				Instruction Type	
NE	EM	TS	MP	FP	WAIT
1	1	0	0	Trap 7	Execute
1	1	1	0	Trap 7	Execute

For the Intel OverDrive Processor, Table 2.5b shows the recommended values for the EM, MP and NE bits.

Table 2.5c shows the interpretation of different combinations of the EM, TS and MP bits for system with an Intel486 SX microprocessor/Intel OverDrive Processor.

Table 2.5b. Recommended Values of the Floating Point Related Bits for a System with an Intel OverDrive™ Processor

CR0 Bit	Intel486 SX Microprocessor	Intel OverDrive Processor
EM	1	0
MP	0	1
NE	1	0, for DOS Systems
		1, for User-Defined Exception Handler

Table 2.5c. Interpretation of Different Combinations of the EM, TS and MP Bits in an Intel486™ SX Microprocessor/Intel OverDrive™ Processor Based System

CR0 Bit			Instruction Type	
EM	TS	MP	Floating Point	Wait
0	0	0	Execute	Execute
0	0	1	Execute	Execute
0	1	0	Exception 7	Execute
0	1	1	Exception 7	Exception 7
1	0	0	Exception 7	Execute
1	0	1	Exception 7	Execute
1	1	0	Exception 7	Execute
1	1	1	Exception 7	Exception 7

NOTE:
If MP=1 and TS=1, the Intel OverDrive Processor will generate a trap 7 so that the system software can save the floating point status of the old task.

All new CR0 bits added to the Intel386 and Intel486 SX microprocessors/Intel OverDrive Processor, except for ET and NE, are upward compatible with the 80286 because they are in register bits not defined in the 80286. For strict compatibility with the 80286, the load machine status word (LMSW) instruction is defined to not change the ET or NE bits.

Control Register 1 (CR1)

CR1 is reserved for use in future Intel microprocessors.

Control Register 2 (CR2)

CR2, shown in Figure 2.6, holds the 32-bit linear address that caused the last page fault detected. The error code pushed onto the page fault handler's stack when it is invoked provides additional status information on this page fault.

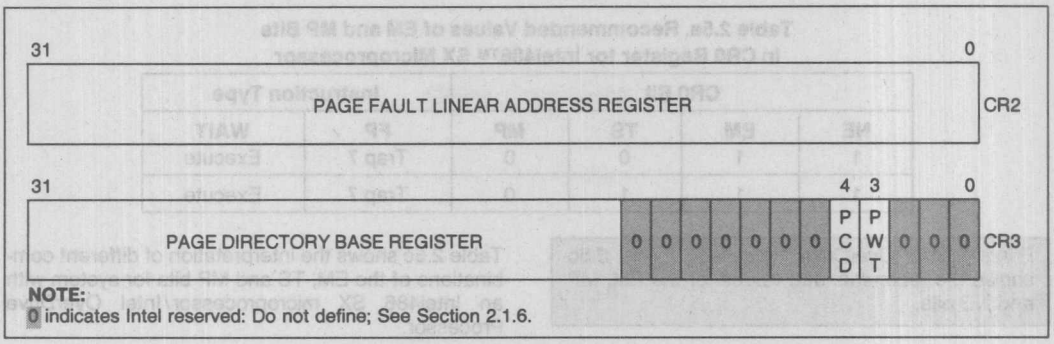


Figure 2.6. Control Registers 2 and 3

Control Register 3 (CR3)

CR3, shown in Figure 2.6, contains the physical base address of the page directory table. The page directory is always page aligned (4 Kbyte-aligned). This alignment is enforced by only storing bits 20–31 in CR3.

In the Intel486 SX microprocessor/Intel OverDrive Processor, CR3 contains two new bits, page write-through (PWT) (bit 3) and page cache disable (PCD) (bit 4). The page table entry (PTE) and page directory entry (PDE) also contain PWT and PCD bits. PWT and PCD control page cacheability. When a page is accessed in external memory, the state of PWT and PCD are driven out on the PWT and PCD pins. The source of PWT and PCD can be CR3, the PTE or the PDE. PWT and PCD are sourced from CR3 when the PDE is being updated. When paging is disabled (PG = 0 in CR0), PCD and PWT are assumed to be 0, regardless of their state in CR3.

A task switch through a task state segment (TSS) which changes the values in CR3, or an explicit load into CR3 with any value, will invalidate all cached page table entries in the translation lookaside buffer (TLB).

The page directory base address in CR3 is a physical address. The page directory can be paged out while its associated task is suspended, but the operating system must ensure that the page directory is resident in physical memory before the task is dispatched. The entry in the TSS for CR3 has a physical address, with no provision for a present bit. This means that the page directory for a task must be resident in physical memory. The CR3 image in a TSS must point to this area, before the task can be dispatched through its TSS.

2.1.2.2 System Address Registers

Four special registers are defined to reference the tables or segments supported by the 80286, Intel386, Intel486 SX microprocessors/Intel OverDrive Processor protection model. These tables or segments are:

GDT (Global Descriptor Table)
IDT (Interrupt Descriptor Table)
LDT (Local Descriptor Table)
TSS (Task State Segment)

The addresses of these tables and segments are stored in special registers, the System Address and System Segment Registers, illustrated in Figure 2.4. These registers are named GDTR, IDTR, LDTR and TR respectively. Section 4, Protected Mode Architecture, describes the use of these registers.

System Address Registers: GDTR and IDTR

The GDTR and IDTR hold the 32-bit linear base address and 16-bit limit of the GDT and IDT, respectively.

Since the GDT and IDT segments are global to all tasks in the system, the GDT and IDT are defined by 32-bit linear addresses (subject to page translation if paging is enabled) and 16-bit limit values.

System Segment Registers: LDTR and TR

The LDTR and TR hold the 16-bit selector for the LDT descriptor and the TSS descriptor, respectively.

Since the LDT and TSS segments are task specific segments, the LDT and TSS are defined by selector values stored in the system segment registers.

NOTE:
A programmer-invisible segment descriptor register is associated with each system segment register.

2.1.3 FLOATING POINT REGISTERS

Figure 2.7 shows the floating point register set. The on-chip FPU contains eight data registers, a tag word, a control register, a status register, an instruction pointer and a data pointer.

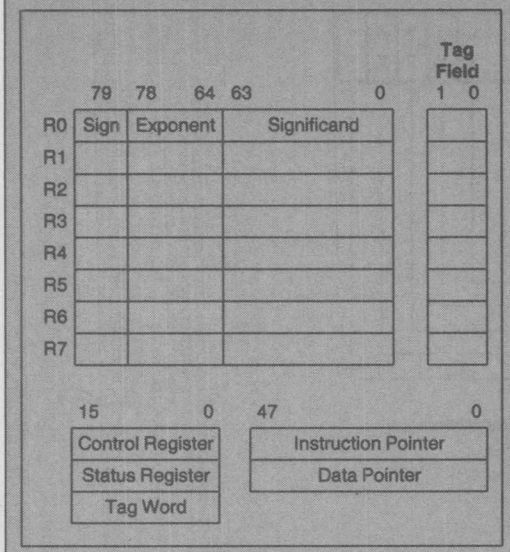


Figure 2.7. Floating Point Registers

The operation of the Intel OverDrive Processor on-chip floating point unit is exactly the same as the Intel387 Math CoProcessor and Intel486 DX microprocessor. Software written for the Intel387 Math CoProcessor will run on the on-chip floating point unit (FPU) without any modifications.

2.1.3.1 Data Registers

Floating point computations use the Intel OverDrive Processor FPU data registers. These eight 80-bit registers provide the equivalent capacity of twenty 32-bit registers. Each of the eight data registers is divided into "fields" corresponding to the FPU's extended-precision data type.

The FPU's register set can be accessed either as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers. The TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by one and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by one. Like other Intel OverDrive Processor stacks in memory, the FPU register stack grows "down" toward lower-addressed registers.

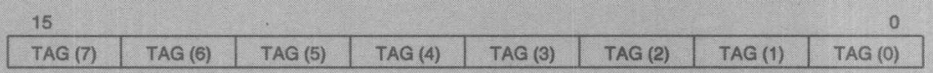
Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to use. This explicit register addressing is also relative to TOP.

2.1.3.2 Tag Word

The tag word marks the content of each numeric data register, as shown in Figure 2.8. Each two-bit tag represents one of the eight data registers. The principal function of the tag word is to optimize the FPUs performance and stack handling by making it possible to distinguish between empty and non-empty register locations. It also enables exception handlers to check the contents of a stack location without the need to perform complex decoding of the actual data.

2.1.3.3 Status Word

The 16-bit status word reflects the overall state of the FPU. The status word is shown in Figure 2.9 and is located in the status register.



NOTE:
The index i of tag(i) is **not** top-relative. A program typically uses the "top" field of Status Word to determine which tag(i) field refers to logical top of stack.
TAG VALUES:
00 = Valid
01 = Zero
10 = QNaN, SNaN, Infinity, Denormal and Unsupported Formats
11 = Empty

Figure 2.8. FPU Tag Word

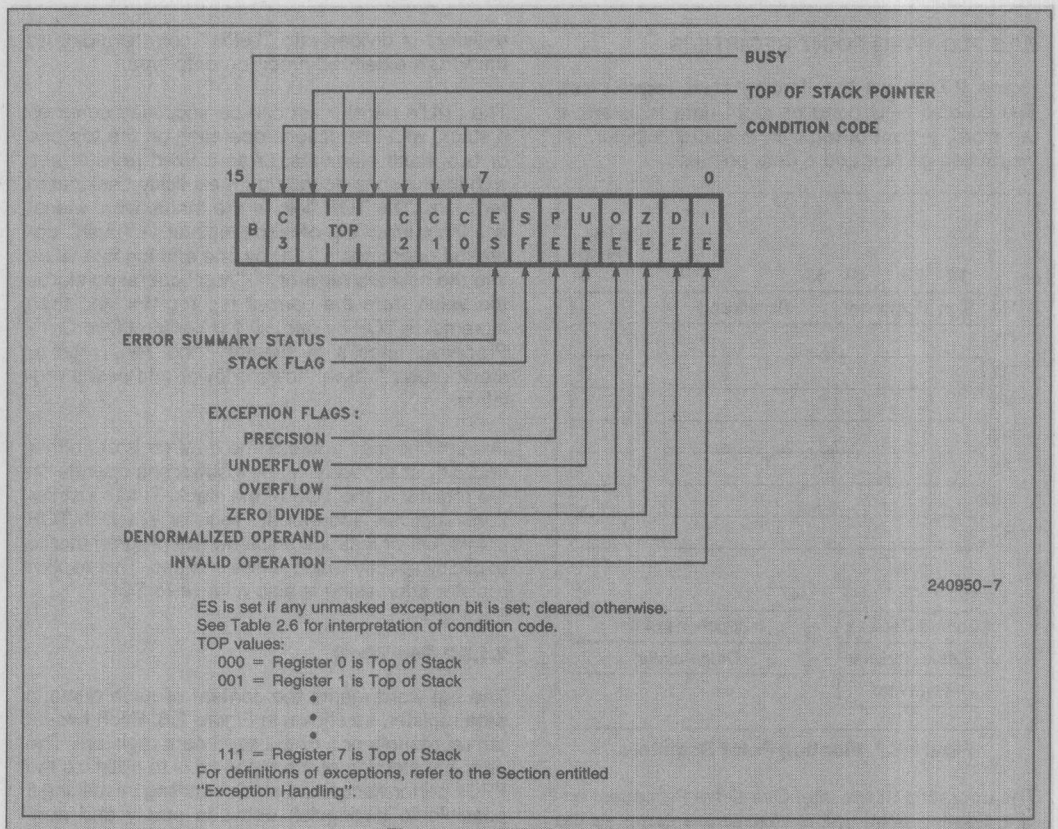


Figure 2.9. FPU Status Word

The B bit (Busy, bit 15) is included for 8087 compatibility. The B bit reflects the contents of the ES bit (bit 7 of the status word).

Bits 13–11 (TOP) point to the FPU register that is the current top-of-stack.

The four numeric condition code bits, C0–C3, are similar to the flags in EFLAGS. Instructions that perform arithmetic operations update C0–C3 to reflect the outcome. The effects of these instructions on the condition codes are summarized in Tables 2.6 through 2.9.

Table 2.6. FPU Condition Code Interpretation

Instruction	C0 (S)	C3 (Z)	C1 (A)	C2 (C)
FPREM, FPREM1 (see Table 2.3)	Three least significant bits of quotient: Q2 / Q0		Q1 or O/U#	Reduction 0 = complete 1 = incomplete
FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of comparison (see Table 2.8)		Zero or O/U#	Operand is not comparable (Table 2.8)
FXAM	Operand class (see Table 2.9)		Sign or O/U#	Operand class (Table 2.9)
FCHS, FABS, FXCH, FINCTOP, FDECTOP, Constant loads, FXTRACT, FLD, FILD, FBLD, FSTP (ext real)	UNDEFINED		Zero or O/U#	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/U#	UNDEFINED
FPTAN, FSIN FCOS, FSINCOS	UNDEFINED		Roundup or O/U#, undefined if C2 = 1	Reduction 0 = complete 1 = incomplete
FLDENV, FRSTOR	Each bit loaded from memory			
FINIT	Clears these bits			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FSAVE	UNDEFINED			

O/U#	When both IE and SF bits of status word are set, indicating a stack exception, this bit distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).
Reduction	If FPREM or FPREM1 produces a remainder that is less than the modulus, reduction is complete. When reduction is incomplete the value at the top of the stack is a partial remainder, which can be used as input to further reduction. For FPTAN, FSIN, FCOS, and FSINCOS, the reduction bit is set if the operand at the top of the stack is too large. In this case the original operand remains at the top of the stack.
Roundup	When the PE bit of the status word is set, this bit indicates whether the last rounding in the instruction was upward.
UNDEFINED	Do not rely on finding any specific value in these bits.

Table 2.7. Condition Code Interpretation after FPREM and FPREM1 Instructions

Condition Code				Interpretation after FPREM and FPREM1	
C2	C3	C1	C0		
1	X	X	X	Incomplete Reduction: further interaction required for complete reduction	
	Q1	Q0	Q2	Q MOD8	Complete Reduction: C0, C3, C1 contain three least significant bits of quotient
0	0	0	0	0	
	0	1	0	1	
	1	0	0	2	
	1	1	0	3	
	0	0	1	4	
	0	1	1	5	
	1	0	1	6	
	1	1	1	7	

Table 2.8. Condition Code Resulting from Comparison

Order	C3	C2	C0
TOP > Operand	0	0	0
TOP < Operand	0	0	1
TOP = Operand	1	0	0
Unordered	1	1	1

Table 2.9. Condition Code Defining Operand Class

C3	C2	C1	C0	Value at TOP
0	0	0	0	+ Unsupported
0	0	0	1	+ NaN
0	0	1	0	− Unsupported
0	0	1	1	− NaN
0	1	0	0	+ Normal
0	1	0	1	+ Infinity
0	1	1	0	− Normal
0	1	1	1	− Infinity
1	0	0	0	+ 0
1	0	0	1	+ Empty
1	0	1	0	− 0
1	0	1	1	− Empty
1	1	0	0	+ Denormal
1	1	1	0	− Denormal

Bit 7 is the error summary (ES) status bit. The ES bit is set if any unmasked exception bit (bits 0–5 in the status word) is set; ES is clear otherwise. The FERR# (floating point error) signal is asserted when ES is set.

Bit 6 is the stack flag (SF). This bit is used to distinguish invalid operations due to stack overflow or underflow. When SF is set, bit 9 (C1) distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).

Table 2.10 shows the six exception flags in bits 0–5 of the status word. Bits 0–5 are set to indicate that the FPU has detected an exception while executing an instruction.

The six exception flags in the status word can be individually masked by mask bits in the FPU control word. Table 2.10 lists the exception conditions, and their causes in order of precedence. Table 2.10 also shows the action taken by the FPU if the corresponding exception flag is masked.

An exception that is not masked by the control word will cause three things to happen: the corresponding exception flag in the status word will be set, the ES bit in the status word will be set and the FERR# output signal will be asserted. When the Intel OverDrive Processor attempts to execute another floating point or WAIT instruction, exception 16 occurs or an external interrupt happens if the NE = 1 in control register 0. The exception condition must be resolved via an interrupt service routine. The FPU saves the address of the floating point instruction that caused the exception and the address of any memory operand required by that instruction in the instruction and data pointers (see Section 2.1.3.4).

Note that when a new value is loaded into the status word by the FLDENV (load environment) or FRSTOR (restore state) instruction, the value of ES (bit 7) and its reflection in the B bit (bit 15) are not derived from the values loaded from memory. The values of ES and B are dependent upon the values of the exception flags in the status word and their corresponding masks in the control word. If ES is set in such a case, the FERR# output of the Intel OverDrive Processor is activated immediately.

2.1.3.4 Instruction and Data Pointers

Because the FPU operates in parallel with the ALU (in the Intel OverDrive Processor the arithmetic and logic unit (ALU) consists of the base architecture registers), any errors detected by the FPU may be reported after the ALU has executed the floating point instruction that caused it. To allow identification of the failing numeric instruction, the Intel OverDrive Processor contains two pointer registers that supply the address of the failing numeric instruction and the address of its numeric memory operand (if appropriate).

Table 2.10. FPU Exceptions

Exception	Cause	Default Action (if exception is masked)
Invalid Operation	Operation on a signaling NaN, unsupported format, indeterminate form ($0*\infty$, 0/0, $(+\infty) + (-\infty)$, etc.), or stack overflow/underflow (SF is also set).	Result is a quiet NaN, integer indefinite, or BCD indefinite
Denormalized Operand	At least one of the operands is denormalized, i.e., it has the smallest exponent but a nonzero significand.	Normal processing continues
Zero Divisor	The divisor is zero while the dividend is a noninfinite, nonzero number.	Result is ∞
Overflow	The result is too large in magnitude to fit in the specified format.	Result is largest finite value or ∞
Underflow	The true result is nonzero but too small to be represented in the specified format, and, if underflow exception is masked, denormalization causes loss of accuracy.	Result is denormalized or zero
Inexact Result (Precision)	The true result is not exactly representable in the specified format (e.g., 1/3); the result is rounded according to the rounding mode.	Normal processing continues

The instruction and data pointers are provided for user-written error handlers. These registers are accessed by the FLDENV (load environment), FSTENV (store environment), FSAVE (save state) and FRSTOR (restore state) instructions. Whenever the Intel OverDrive Processor decodes a new floating point instruction, it saves the instruction (including any prefixes that may be present), the address of the operand (if present) and the opcode.

The instruction and data pointers appear in one of four formats depending on the operating mode of the Intel OverDrive Processor (protected mode or real-address mode) and depending on the operand-size attribute in effect (32-bit operand or 16-bit operand). When the Intel OverDrive Processor is in the virtual-86 mode, the real address mode formats are used. The four formats are shown in Figures 2.10–2.13. The floating point instructions FLDENV, FSTENV, FSAVE and FRSTOR are used to transfer these values to and from memory. Note that the value of the data pointer is undefined if the prior floating point instruction did not have a memory operand.

NOTE:
The operand size attribute is the D bit in a segment descriptor.

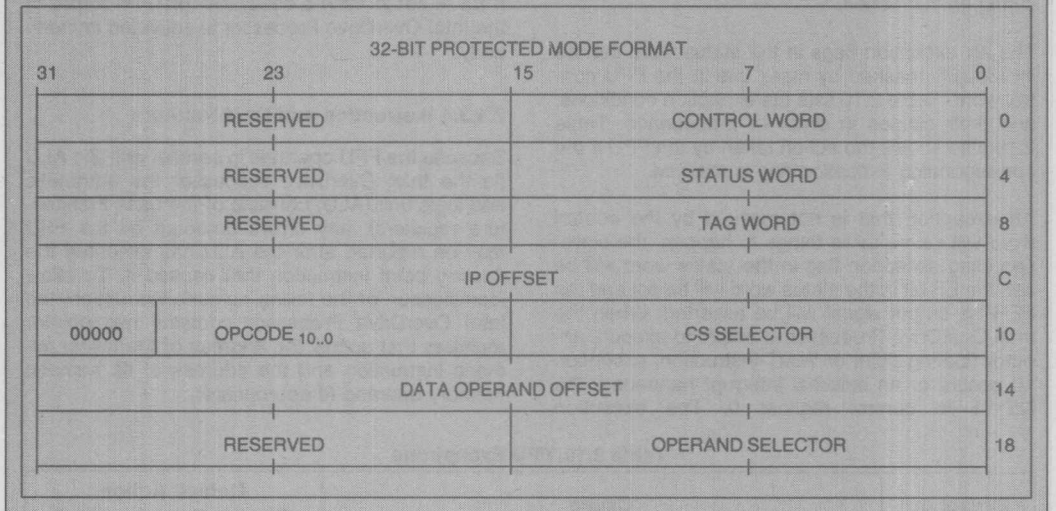


Figure 2.10. Protected Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

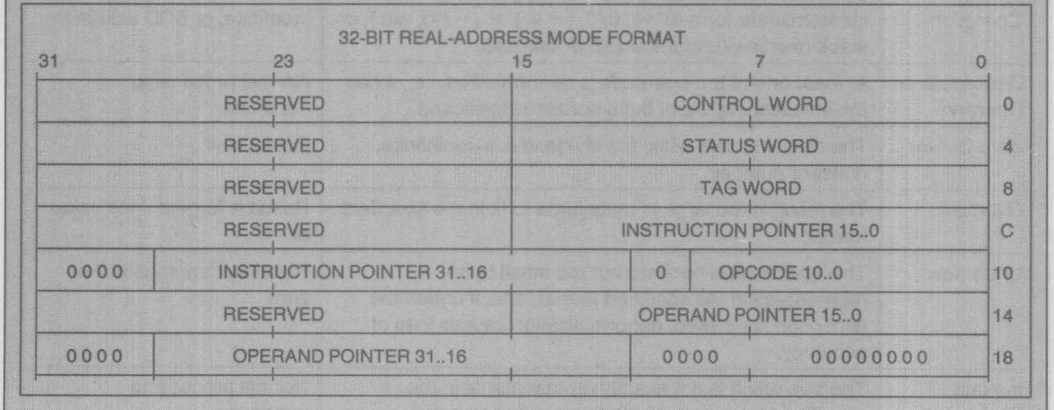


Figure 2.11. Real Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

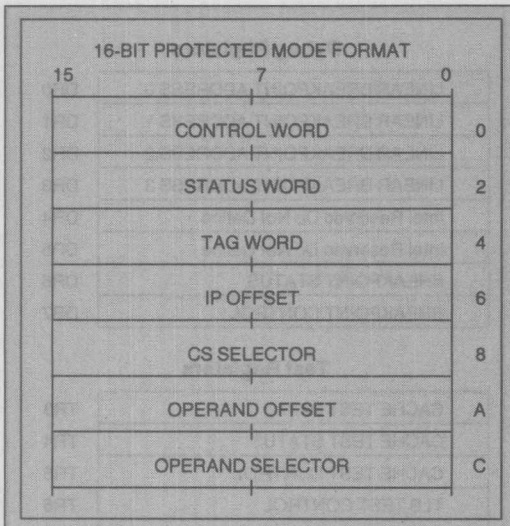


Figure 2.12. Protected Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

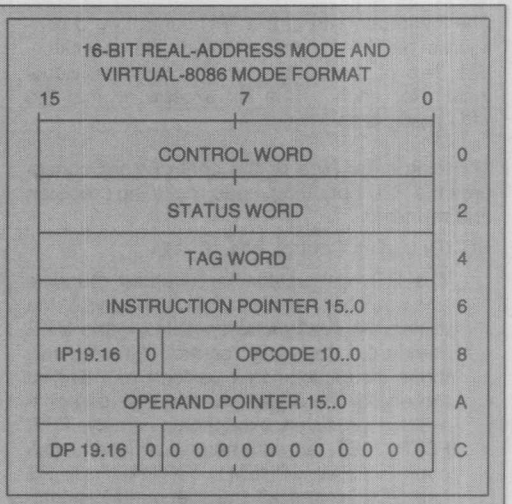
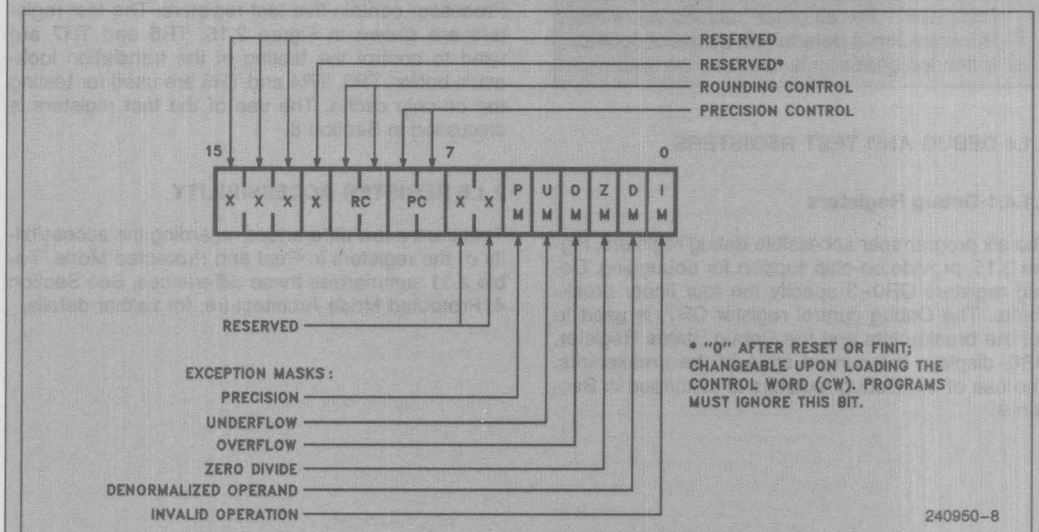


Figure 2.13. Real Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

2

2.1.3.5 FPU Control Word

The FPU provides several processing options that are selected by loading a control word from memory into the control register. Figure 2.14 shows the format and encoding of fields in the control word.



Precision Control
00—24 bits (single precision)
01—(reserved)
10—53 bits (double precision)
11—64 bits (extended precision)

Rounding Control
00—Round to nearest or even
01—Round down (toward $-\infty$)
10—Round up (toward $+\infty$)
11—Chop (truncate toward zero)

Figure 2.14. FPU Control Word

The low-order byte of the FPU control word configures the FPU error and exception masking. Bits 0–5 of the control word contain individual masks for each of the six exceptions that the FPU recognizes.

The high-order byte of the control word configures the FPU operating mode, including precision and rounding.

RC (Rounding Control, bits 10–11)

The RC bits provide for directed rounding and true chop, as well as the unbiased round to nearest even mode specified in the IEEE standard. Rounding control affects only those instructions that perform rounding at the end of the operation (and thus can generate a precision exception); namely, FST, FSTP, FIST, all arithmetic instructions (except FPREM, FPREM1, FXTRACT, FABS and FCHS), and all transcendental instructions.

PC (Precision Control, bits 8–9)

The PC bits can be used to set the FPU internal operating precision of the significand at less than the default of 64 bits (extended precision). This can be useful in providing compatibility with early generation arithmetic processors of smaller precision. PC affects only the instructions ADD, SUB, DIV, MUL, and SQRT. For all other instructions, either the precision is determined by the opcode or extended precision is used.

2.1.4 DEBUG AND TEST REGISTERS

2.1.4.1 Debug Registers

The six programmer accessible debug registers, Figure 2.15, provide on-chip support for debugging. Debug registers DR0–3 specify the four linear breakpoints. The Debug control register DR7, is used to set the breakpoints and the Debug Status Register, DR6, displays the current state of the breakpoints. The use of the Debug registers is described in Section 9.

Debug Registers

LINEAR BREAKPOINT ADDRESS 0	DR0
LINEAR BREAKPOINT ADDRESS 1	DR1
LINEAR BREAKPOINT ADDRESS 2	DR2
LINEAR BREAKPOINT ADDRESS 3	DR3
Intel Reserved Do Not Define	DR4
Intel Reserved Do Not Define	DR5
BREAKPOINT STATUS	DR6
BREAKPOINT CONTROL	DR7

Test Registers

CACHE TEST DATA	TR3
CACHE TEST STATUS	TR4
CACHE TEST CONTROL	TR5
TLB TEST CONTROL	TR6
TLB TEST STATUS	TR7

TLB = Translation Lookaside Buffer

Figure 2.15

2.1.4.2 Test Registers

The Intel486 SX microprocessor/Intel OverDrive Processor contain five test registers. The test registers are shown in Figure 2.15. TR6 and TR7 are used to control the testing of the translation lookaside buffer. TR3, TR4 and TR5 are used for testing the on-chip cache. The use of the test registers is discussed in Section 8.

2.1.5 REGISTER ACCESSIBILITY

There are a few differences regarding the accessibility of the registers in Real and Protected Mode. Table 2.11 summarizes these differences. See Section 4, Protected Mode Architecture, for further details.

Table 2.11. Register Usage

Register	Use in Real Mode		Use in Protected Mode		Use in Virtual 8086 Mode	
	Load	Store	Load	Store	Load	Store
General Registers	Yes	Yes	Yes	Yes	Yes	Yes
Segment Register	Yes	Yes	Yes	Yes	Yes	Yes
Flag Register	Yes	Yes	Yes	Yes	IOPL	IOPL*
Control Registers	Yes	Yes	PL = 0	PL = 0	No	Yes
GDTR	Yes	Yes	PL = 0	Yes	No	Yes
IDTR	Yes	Yes	PL = 0	Yes	No	Yes
LDTR	No	No	PL = 0	Yes	No	No
TR	No	No	PL = 0	Yes	No	No
FPU Data Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Control Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Status Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Instruction Pointer	Yes	Yes	Yes	Yes	Yes	Yes
FPU Data Pointer	Yes	Yes	Yes	Yes	Yes	Yes
Debug Registers	Yes	Yes	PL = 0	PL = 0	No	No
Test Registers	Yes	Yes	PL = 0	PL = 0	No	No

NOTES:
PL = 0: The registers can be accessed only when the current privilege level is zero.
*IOPL: The PUSHF and POPF instructions are made I/O Privilege Level sensitive in Virtual 86 Mode.

2.1.6 COMPATIBILITY

VERY IMPORTANT NOTE:
COMPATIBILITY WITH FUTURE PROCESSORS

In the preceding register descriptions, note certain Intel486 SX microprocessor/Intel OverDrive™ Processor register bits are Intel reserved. When reserved bits are called out, treat them as fully undefined. This is essential for your software compatibility with future processors! Follow the guidelines below:

1. **Do not depend on the states of any undefined bits when testing the values of defined register bits. Mask them out when testing.**
2. **Do not depend on the states of any undefined bits when storing them to memory or another register.**
3. **Do not depend on the ability to retain information written into any undefined bits.**
4. **When loading registers always load the undefined bits as zeros.**
5. **However, registers which have been previously stored may be reloaded without masking.**

Depending upon the values of undefined register bits will make your software dependent upon the unspecified Intel486 SX microprocessor/Intel OverDrive Processor handling of these bits. Depending on undefined values risks making your software incompatible with future processors that define usages for the Intel486 SX microprocessor-undefined bits and Intel OverDrive Processor bits. AVOID ANY SOFTWARE DEPENDENCE UPON THE STATE OF UNDEFINED Intel486 SX MICROPROCESSOR/Intel OverDrive™ PROCESSOR REGISTER BITS.

2.2 Instruction Set

The Intel486 SX microprocessor/Intel OverDrive Processor instruction set can be divided into 11 categories of operations:

- Data Transfer
- Arithmetic
- Shift/Rotate
- String Manipulation
- Bit Manipulation
- Control Transfer
- High Level Language Support
- Operating System Support
- Processor Control

- Floating Point
- Floating Point Control

The Intel486 SX microprocessor/Intel OverDrive Processor instructions are listed in Section 10.

Only the Intel OverDrive Processor has floating point instructions. The Intel486 SX microprocessor does not have floating point instructions. Note that all floating point unit instruction mnemonics begin with an F.

All Intel486 SX microprocessor/Intel OverDrive Processor instructions operate on either 0, 1, 2 or 3 operands; where an operand resides in a register, in the instruction itself or in memory. Most zero operand instructions (e.g., CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2 bytes long. Since the Intel486 SX microprocessor/Intel OverDrive Processor have a 32-byte instruction queue, an average of 10 instructions will be prefetched. The use of two operands permits the following types of common instructions:

- Register to Register
- Memory to Register
- Memory to Memory
- Immediate to Register
- Register to Memory
- Immediate to Memory

The operands can be either 8, 16, or 32 bits long. As a general rule, when executing 32-bit code, operands are 8 or 32 bits; when executing existing 80286 or 8086 code (16-bit code), operands are 8 or 16 bits. Prefixes can be added to all instructions which override the default length of the operands (i.e., use 32-bit operands for 16-bit code, or 16-bit operands for 32-bit code).

2.3 Memory Organization

Introduction

Memory on the Intel486 SX Microprocessor/Intel OverDrive Processor is divided up into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address, the high order byte at the high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address, the high-order byte at the highest address. The address of a word or dword is the byte address of the low-order byte.

In addition to these basic data types, the Intel486 SX microprocessor/Intel OverDrive Processor support two larger units of memory: pages and segments. Memory can be divided up into one or more variable length segments, which can be swapped to disk or shared between programs. Memory can also be organized into one or more 4 Kbyte pages. Finally, both segmentation and paging can be combined, gaining the advantages of both systems. The Intel486 SX microprocessor/Intel OverDrive Processor support both pages and segments in order to provide maximum flexibility to the system designer. Segmentation and paging are complementary. Segmentation is useful for organizing memory in logical modules, and as such is a tool for the application programmer, while pages are useful for the system programmer for managing the physical memory of a system.

2.3.1 ADDRESS SPACES

The Intel486 SX microprocessor/Intel OverDrive Processor have three distinct address spaces: **logical, linear,** and **physical**. A **logical** address (also known as a **virtual** address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, DISPLACEMENT) discussed in Section 2.5.3 **Memory Addressing Modes** into an effective address. Since each task on the Intel486 SX microprocessor/Intel OverDrive Processor have a maximum of 16K ($2^{14}-1$) selectors, and offsets can be 4 gigabytes, (2^{32} bits) this gives a total of 2^{46} bits or 64 terabytes of **logical** address space per task. The programmer sees this virtual address space.

The segmentation unit translates the **logical** address space into a 32-bit **linear** address space. If the paging unit is not enabled then the 32-bit **linear** address corresponds to the **physical** address. The paging unit translates the **linear** address space into the **physical** address space. The **physical address** is what appears on the address pins.

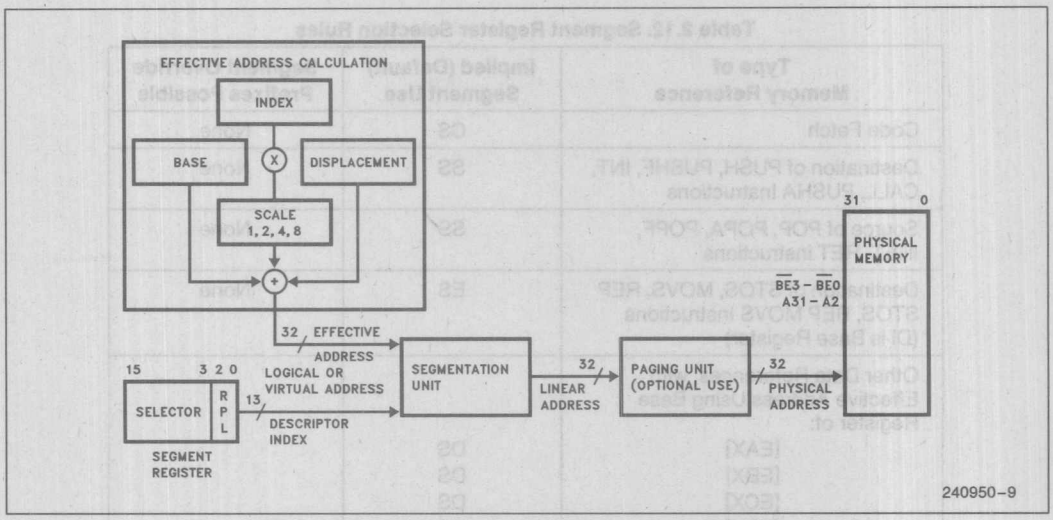


Figure 2.16. Address Translation

The primary difference between Real Mode and Protected Mode is how the segmentation unit performs the translation of the **logical** address into the **linear** address. In Real Mode, the segmentation unit shifts the selector left four bits and adds the result to the offset to form the **linear** address. While in Protected Mode every selector has a **linear** base address associated with it. The **linear base** address is stored in one of two operating system tables (i.e., the Local Descriptor Table or Global Descriptor Table). The selector's **linear base** address is added to the offset to form the final **linear** address.

Figure 2.16 shows the relationship between the various address spaces.

2.3.2 SEGMENT REGISTER USAGE

The main data structure used to organize memory is the segment. On the Intel486 SX microprocessor/Intel OverDrive Processor, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments: code and data, the segments are of variable size and can be as small as 1 byte or as large as 4 gigabytes (2^{32} bytes).

In order to provide compact instruction encoding, and increase Intel486 SX microprocessor/Intel OverDrive Processor performance, instructions do not need to explicitly specify which segment register is used. A default segment register is automatically chosen according to the rules of Table 2.12 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register; Stack references use the SS register and Instruction fetches use the CS register. The contents of the Instruction Pointer provide the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 2.12. The override prefixes also allow the use of the ES, FS and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero and create a system with a four gigabyte linear address space. This creates a system where the virtual address space is the same as the linear address space. Further details of segmentation are discussed in Section 4.1.

2.4 I/O Space

The Intel486 SX microprocessor/Intel OverDrive Processor have two distinct physical address spaces: Memory and I/O. Generally, peripherals are placed in I/O space although the Intel486 SX microprocessor/Intel OverDrive Processor also supports memory-mapped peripherals. The I/O space consists of 64 Kbytes, it can be divided into 64K 8-bit ports, 32K 16-bit ports, or 16K 32-bit ports, or any combination of ports which add up to less than 64 Kbytes. The 64K I/O address space refers to physical memory rather than linear address since I/O instructions do not go through the segmentation or paging hardware. The M/IO# pin acts as an additional address line thus allowing the system designer to easily determine which address space the processor is accessing.

Table 2.12. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment Use	Segment Override Prefixes Possible
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA Instructions	SS	None
Source of POP, POPA, POPF, IRET, RET instructions	SS	None
Destination of STOS, MOVS, REP STOS, REP MOVS Instructions (DI is Base Register)	ES	None
Other Data References, with Effective Address Using Base Register of:		
[EAX]	DS	All
[EBX]	DS	
[ECX]	DS	
[EDX]	DS	
[ESI]	DS	
[EDI]	DS	
[EBP]	SS	
[ESP]	SS	

The I/O ports are accessed via the IN and OUT I/O instructions, with the port address supplied as an immediate 8-bit constant in the instruction or in the DX register. All 8- and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/IO# pin to be driven low.

I/O port addresses 00F8H through 00FFH are reserved for use by Intel.

2.5 Addressing Modes

2.5.1 ADDRESSING MODES OVERVIEW

The Intel486 SX microprocessor/Intel OverDrive Processor provide a total of 11 addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

2.5.2 REGISTER AND IMMEDIATE MODES

Two of the addressing modes provide for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8-, 16- or 32-bit general registers.

Immediate Operand Mode: The operand is included in the instruction as part of the opcode.

2.5.3 32-BIT MEMORY ADDRESSING MODES

The remaining 9 modes provide a mechanism for specifying the effective address of an operand. The linear address consists of two components: the segment base address and an effective address. The effective address is calculated by using combinations of the following four address elements:

DISPLACEMENT: An 8-, or 32-bit immediate value, following the instruction.

BASE: The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area.

INDEX: The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array, or a string of characters.

SCALE: The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. Scaled index mode is especially useful for accessing arrays or structures.

Combinations of these 4 components make up the 9 additional addressing modes. There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions. The one exception is the simultaneous use of Base and Index components which requires one additional clock.

As shown in Figure 2.17, the effective address (EA) of an operand is calculated according to the following formula.

EA = Base Reg + (Index Reg * Scaling) + Displacement

Direct Mode: The operand's offset is contained as part of the instruction as an 8-, 16- or 32-bit displacement.
EXAMPLE: INC Word PTR [500]

Register Indirect Mode: A BASE register contains the address of the operand.
EXAMPLE: MOV [ECX], EDX

Based Mode: A BASE register's contents is added to a DISPLACEMENT to form the operand's offset.
EXAMPLE: MOV ECX, [EAX + 24]

Index Mode: An INDEX register's contents is added to a DISPLACEMENT to form the operand's offset.
EXAMPLE: ADD EAX, TABLE[ESI]

Scaled Index Mode: An INDEX register's contents is multiplied by a scaling factor which is added to a DISPLACEMENT to form the operand's offset.
EXAMPLE: IMUL EBX, TABLE[ESI*4],7

Based Index Mode: The contents of a BASE register is added to the contents of an INDEX register to form the effective address of an operand.
EXAMPLE: MOV EAX, [ESI] [EBX]

Based Scaled Index Mode: The contents of an INDEX register is multiplied by a SCALING factor and the result is added to the contents of a BASE register to obtain the operand's offset.
EXAMPLE: MOV ECX, [EDX*8] [EAX]

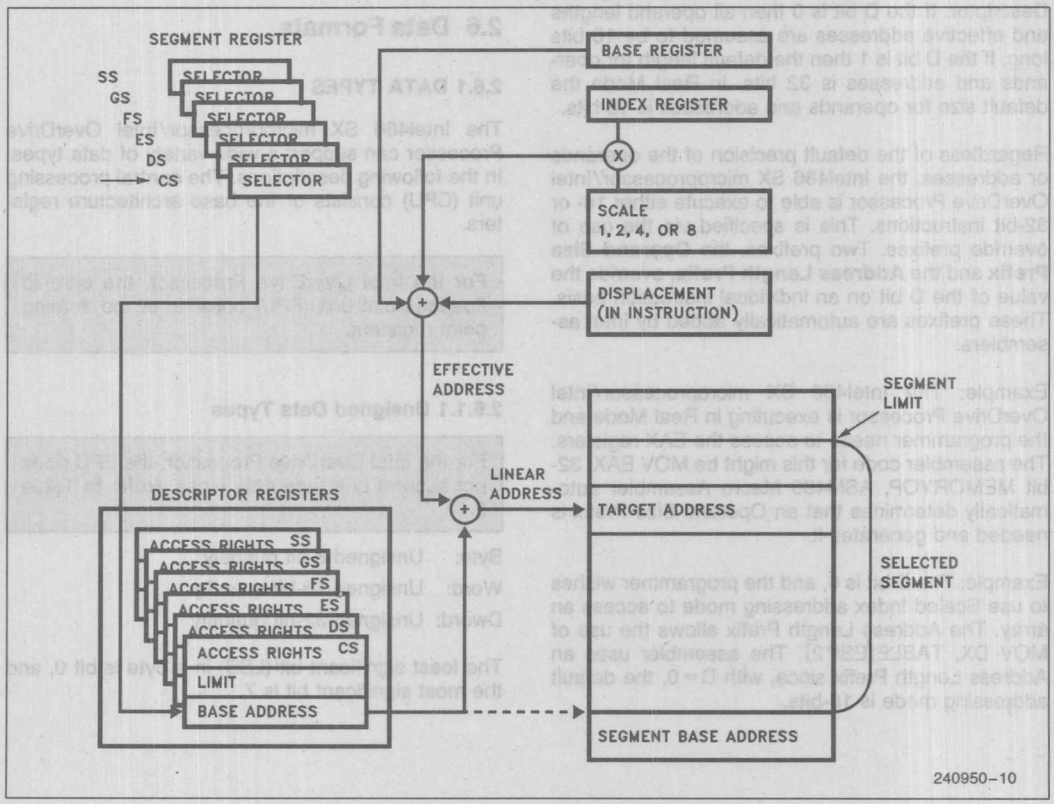


Figure 2.17. Addressing Mode Calculations

Based Index Mode with Displacement: The contents of an INDEX Register and a BASE register's contents and a DISPLACEMENT are all summed together to form the operand offset.
EXAMPLE: ADD EDX, [ESI] [EBP+00FFFFF0H]

Based Scaled Index Mode with Displacement: The contents of an INDEX register are multiplied by a SCALING factor, the result is added to the contents of a BASE register and a DISPLACEMENT to form the operand's offset.
EXAMPLE: MOV EAX, LOCALTABLE[EDI*4] [EBP+80]

2.5.4 DIFFERENCES BETWEEN 16- AND 32-BIT ADDRESSES

In order to provide software compatibility with the 80286 and the 8086, the Intel486 SX microprocessor/Intel OverDrive Processor can execute 16-bit instructions in Real and Protected Modes. The processor determines the size of the instructions it is executing by examining the D bit in the CS segment Descriptor. If the D bit is 0 then all operand lengths and effective addresses are assumed to be 16 bits long. If the D bit is 1 then the default length for operands and addresses is 32 bits. In Real Mode the default size for operands and addresses is 16-bits.

Regardless of the default precision of the operands or addresses, the Intel486 SX microprocessor/Intel OverDrive Processor is able to execute either 16- or 32-bit instructions. This is specified via the use of override prefixes. Two prefixes, the **Operand Size Prefix** and the **Address Length Prefix**, override the value of the D bit on an individual instruction basis. These prefixes are automatically added by Intel assemblers.

Example: The Intel486 SX microprocessor/Intel OverDrive Processor is executing in Real Mode and the programmer needs to access the EAX registers. The assembler code for this might be MOV EAX, 32-bit MEMORYOP, ASM486 Macro Assembler automatically determines that an Operand Size Prefix is needed and generates it.

Example: The D bit is 0, and the programmer wishes to use Scaled Index addressing mode to access an array. The Address Length Prefix allows the use of MOV DX, TABLE[ESI*2]. The assembler uses an Address Length Prefix since, with D=0, the default addressing mode is 16-bits.

Example: The D bit is 1, and the program wants to store a 16-bit quantity. The Operand Length Prefix is used to specify only a 16-bit value; MOV MEM16, DX.

The OPERAND LENGTH and Address Length Prefixes can be applied separately or in combination to any instruction. The Address Length Prefix does not allow addresses over 64 Kbytes to be accessed in Real Mode. A memory address which exceeds FFFFH will result in a General Protection Fault. An Address Length Prefix only allows the use of the additional Intel486 SX microprocessor/Intel OverDrive Processor addressing modes.

When executing 32-bit code, the Intel486 SX microprocessor/Intel OverDrive Processor uses either 8-, or 32-bit displacements, and any register can be used as base or index registers. When executing 16-bit code, the displacements are either 8, or 16 bits, and the base and index register conform to the 80286 model. Table 2.13 illustrates the differences.

2.6 Data Formats

2.6.1 DATA TYPES

The Intel486 SX microprocessor/Intel OverDrive Processor can support a wide-variety of data types. In the following descriptions. The central processing unit (CPU) consists of the base architecture registers.

For the Intel OverDrive Processor, the on-chip floating point unit (FPU) consists of the floating point registers.

2.6.1.1 Unsigned Data Types

For the Intel OverDrive Processor, the FPU does not support unsigned data types. Refer to Table 2.14.

Byte: Unsigned 8-bit quantity
Word: Unsigned 16-bit quantity
Dword: Unsigned 32-bit quantity

The least significant bit (LSB) in a byte is bit 0, and the most significant bit is 7.

Table 2.13. BASE and INDEX Registers for 16- and 32-Bit Addresses

	16-Bit Addressing	32-Bit Addressing
BASE REGISTER	BX,BP	Any 32-bit GP Register
INDEX REGISTER	SI,DI	Any 32-bit GP Register Except ESP
SCALE FACTOR	none	1, 2, 4, 8
DISPLACEMENT	0, 8, 16 bits	0, 8, 32 bits

2.6.1.2 Signed Data Types

All signed data types assume 2's complement notation. The signed data types contain two fields, a sign bit and a magnitude. The sign bit is the most significant bit (MSB). The number is negative if the sign bit is 1. If the sign bit is 0, the number is positive. The magnitude field consists of the remaining bits in the number. Refer to Table 2.14.

8-bit Integer: Signed 8-bit quantity

16-bit Integer: Signed 16-bit quantity

32-bit Integer: Signed 32-bit quantity

64-bit Integer: Signed 64-bit quantity

For the Intel OverDrive Processor, the FPU only supports 16-, 32- and 64-bit integers.

The CPU section of the Intel486 SX microprocessor/Intel OverDrive Processor only supports 8-, 16- and 32-bit integers.

2.6.1.3 Floating Point Data Types

Floating point data type in the Intel OverDrive Processor contain three fields, sign, significand and exponent. The sign field is one bit and is the MSB of the floating point number. The number is negative if the sign bit is 1. If the sign bit is 0, the number is positive. The significand gives the significant bits of the number. The exponent field contains the power of 2 needed to scale the significand. Refer to Table 2.14.

Only the FPU supports floating point data types.

Single Precision Real: 23-bit significand and 8-bit exponent. 32 bits total.

Double Precision Real: 52-bit significand and 11-bit exponent. 64 bits total.

Extended Precision Real: 64-bit significand and 15-bit exponent. 80 bits total.

2.6.1.4 BCD Data Types

The Intel486 SX microprocessor/Intel OverDrive Processor support packed and unpacked binary coded decimal (BCD) data types. A packed BCD data type contains two digits per byte, the lower digit is in bits 0–3 and the upper digit in bits 4–7. An unpacked BCD data type contains 1 digit per byte stored in bits 0–3.

The CPU section of the Intel486 SX microprocessor/Intel OverDrive Processor supports 8-bit packed and unpacked BCD data types. Refer to Table 2.14.

For the Intel OverDrive Processor the FPU only supports 80-bit packed BCD data types.

2.6.1.5 String Data Types

A string data type is a contiguous sequence of bits, bytes, words or dwords. A string may contain between 1 byte and 4 Gbytes. Refer to Table 2.15.

String data types are only supported by the CPU section of the Intel486 SX microprocessor/Intel OverDrive Processor.

Byte String: Contiguous sequence of bytes.

Word String: Contiguous sequence of words.

Dword String: Contiguous sequence of dwords.

Bit String: A set of contiguous bits. In the Intel486 SX microprocessor/Intel OverDrive Processor bit strings can be up to 4 gigabits long.

2.6.1.6 ASCII Data Types

The Intel486 SX microprocessor/Intel OverDrive Processor support ASCII (American Standard Code for Information Interchange) strings and can perform arithmetic operations (such as addition and division) on ASCII data. The CPU section of the Intel486 SX microprocessor/Intel OverDrive Processor can only operate on ASCII data. Refer to Table 2.15.

Table 2.14. Intel486™ SX Microprocessor/Intel OverDrive Processor Data Types

Data Format	Supported by Base Registers	Supported by FPU	Range	Precision
Byte	X		0–255	8 bits
Word	X		0–64K	16 bits
Dword	X		0–4G	32 bits
8-Bit Integer	X		10^2	8 bits
16-Bit Integer	X	X	10^4	16 bits
32-Bit Integer	X	X	10^9	32 bits
64-Bit Integer		X	10^{19}	64 bits
8-Bit Unpacked BCD	X		0–9	1 Digit
8-Bit Packed BCD	X		0–9	2 Digits
80-Bit Packed BCD		X	$\pm 10^{\pm 18}$	18 Digits
Single Precision Real		X	$\pm 10^{\pm 38}$	24 Bits
Double Precision Real		X	$\pm 10^{\pm 308}$	53 Bits
Extended Precision Real		X	$\pm 10^{\pm 4932}$	64 Bits

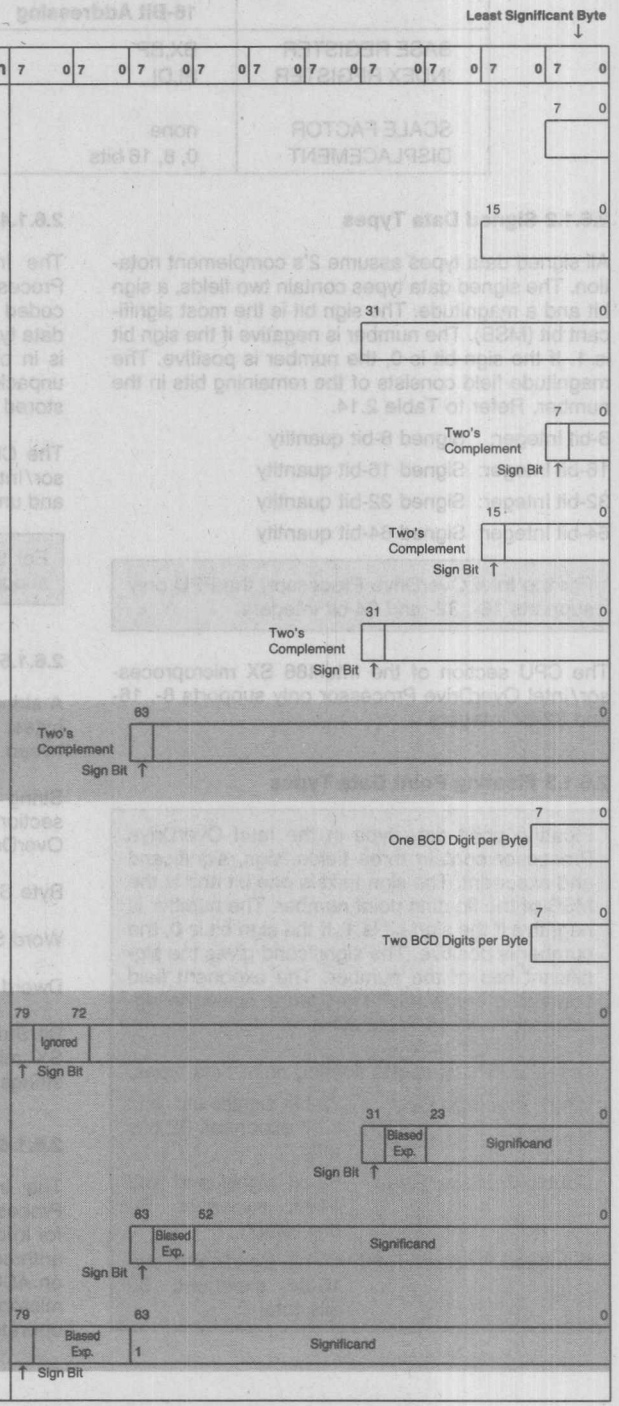
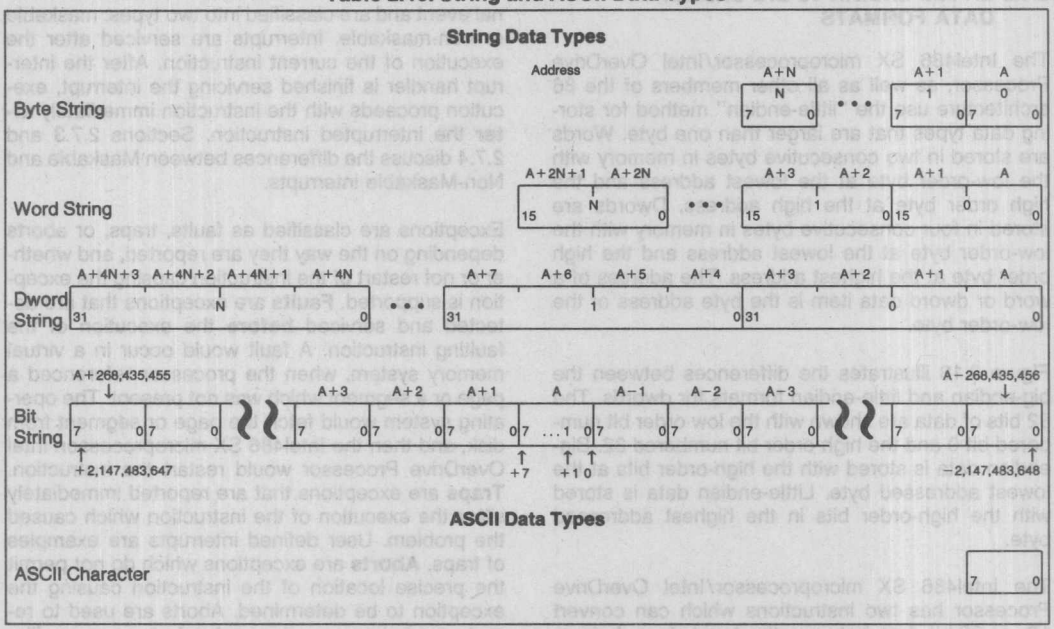


Table 2.15. String and ASCII Data Types



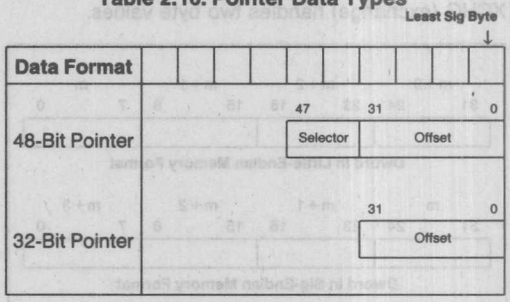
2.6.1.7 Pointer Data Types

A pointer data type contains a value that gives the address of a piece of data. The Intel486 SX microprocessor/Intel OverDrive Processor support two types of pointers. Refer to Table 2.16.

48-bit Pointer: 16-bit selector and 32-bit offset

32-bit Pointer: 32-bit offset

Table 2.16. Pointer Data Types



2.6.2 LITTLE ENDIAN vs BIG ENDIAN DATA FORMATS

The Intel486 SX microprocessor/Intel OverDrive Processor, as well as all other members of the 86 architecture use the "little-endian" method for storing data types that are larger than one byte. Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address and the high order byte at the high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address and the high order byte at the highest address. The address of a word or dword data item is the byte address of the low-order byte.

Figure 2.18 illustrates the differences between the big-endian and little-endian formats for dwords. The 32 bits of data are shown with the low order bit numbered bit 0 and the high order bit numbered 32. Big-endian data is stored with the high-order bits at the lowest addressed byte. Little-endian data is stored with the high-order bits in the highest addressed byte.

The Intel486 SX microprocessor/Intel OverDrive Processor has two instructions which can convert 16- or 32-bit data between the two byte orderings. BSWAP (byte swap) handles four byte values and XCHG (exchange) handles two byte values.

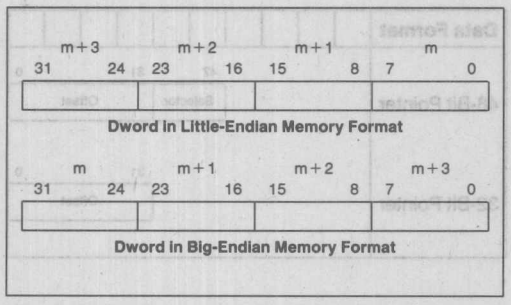


Figure 2.18. Big vs Little Endian Memory Format

2.7 Interrupts

2.7.1 INTERRUPTS AND EXCEPTIONS

Interrupts and exceptions alter the normal program flow, in order to handle external events, to report errors or exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT N instruction, the Intel486 SX microprocessor/Intel OverDrive Processor treats software interrupts as exceptions.

Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately **after** the interrupted instruction. Sections 2.7.3 and 2.7.4 discuss the differences between Maskable and Non-Maskable interrupts.

Exceptions are classified as faults, traps, or aborts depending on the way they are reported, and whether or not restart of the instruction causing the exception is supported. **Faults** are exceptions that are detected and serviced **before** the execution of the faulting instruction. A fault would occur in a virtual memory system, when the processor referenced a page or a segment which was not present. The operating system would fetch the page or segment from disk, and then the Intel486 SX microprocessor/Intel OverDrive Processor would restart the instruction. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. User defined interrupts are examples of traps. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined. Aborts are used to report severe errors, such as a hardware error, or illegal values in system tables.

Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand, the return address from an exception fault routine will always point at the instruction causing the exception and include any leading instruction prefixes. Table 2.17 summarizes the possible interrupts for the Intel486 SX microprocessor/Intel OverDrive Processor and shows where the return address points.

The Intel486 SX microprocessor/Intel OverDrive Processor have the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. In Real Mode (see Section 3.1), the vectors are 4 byte quantities, a Code Segment plus a 16-bit offset; in Protected Mode, the interrupt vectors are 8 byte quantities, which are put in an Interrupt Descriptor Table (see Section 4.3.3.4). Of the 256 possible interrupts, 32 are reserved for use by Intel, the remaining 224 are free to be used by the system designer.

2.7.2 INTERRUPT PROCESSING

When an interrupt occurs the following actions happen. First, the current program address and the

Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the Intel486 SX microprocessor/Intel OverDrive Processor which identifies the appropriate entry in the interrupt table. The table contains the starting address of the interrupt service routine. Then, the user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old Intel486 SX microprocessor/Intel OverDrive Processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the Intel486 SX microprocessor/Intel OverDrive Processor in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

2.7.3 MASKABLE INTERRUPT

Maskable interrupts are the most common way used by the Intel486 SX microprocessor/Intel OverDrive Processor to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled high and the Interrupt Flag bit (IF) is enabled. The Intel486 SX microprocessor/Intel OverDrive Processor only responds to interrupts between instructions, (REPeat String instructions, have an "interrupt window", between memory moves, which allows interrupts during long string moves). When an interrupt occurs the Intel486 SX microprocessor/Intel OverDrive Processor CoProcessor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt, (one of 224 user defined interrupts). The exact nature of the interrupt sequence is discussed in Section 7.2.10.

Table 2.17. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	YES	FAULT
Debug Exception	1	Any Instruction	YES	TRAP*
NMI Interrupt	2	INT 2 or NMI	NO	NMI
One Byte Interrupt	3	INT	NO	TRAP
Interrupt on Overflow	4	INTO	NO	TRAP
Array Bounds Check	5	BOUND	YES	FAULT
Invalid OP-Code	6	Any Illegal Instruction	YES	FAULT
Device Not Available	7	ESC, WAIT	YES	FAULT
Double Fault	8	Any Instruction That Can Generate an Exception		ABORT
Intel Reserved	9			
Invalid TSS	10	JMP, CALL, IRET, INT	YES	FAULT
Segment Not Present	11	Segment Register Instructions	YES	FAULT
Stack Fault	12	Stack References	YES	FAULT
General Protection Fault	13	Any Memory Reference	YES	FAULT
Page Fault	14	Any Memory Access or Code Fetch	YES	FAULT
Intel Reserved	15			
Floating Point Error	16	Floating Point, WAIT	YES	FAULT
Alignment Check Interrupt	17	Unaligned Memory Access	YES	FAULT
Intel Reserved	18–31			
Two Byte Interrupt	0–255	INT n	NO	TRAP

*Some debug exceptions may report both traps on the previous instruction, and faults on the next instruction.

The IF bit in the EFLAG registers is reset when an interrupt is being serviced. This effectively disables servicing additional interrupts during an interrupt service routine. However, the IF may be set explicitly by the interrupt handler, to allow the nesting of interrupts. When an IRET instruction is executed the original state of the IF is restored.

2.7.4 NON-MASKABLE INTERRUPT

Non-maskable interrupts provide a method of servicing very high priority interrupts. A common example of the use of a non-maskable interrupt (NMI) would be to activate a power failure routine. When the NMI input is pulled high it causes an interrupt with an internally supplied vector value of 2. Unlike a normal hardware interrupt, no interrupt acknowledgment sequence is performed for an NMI.

While executing the NMI servicing procedure, the Intel486 SX microprocessor/Intel OverDrive Processor will not service further NMI requests until an interrupt return (IRET) instruction is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The IF bit is cleared at the beginning of an NMI interrupt to inhibit further INTR interrupts.

2.7.5 SOFTWARE INTERRUPTS

A third type of interrupt/exception for the Intel486 SX microprocessor/Intel OverDrive Processor is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table.

A special case of the two byte software interrupt INT n is the one byte INT 3, or breakpoint interrupt. By inserting this one byte instruction in a program, the user can set breakpoints in his program as a debugging tool.

A final type of software interrupt is the single step interrupt. It is discussed in Section 9.2.

2.7.6 INTERRUPT AND EXCEPTION PRIORITIES

Interrupts are externally-generated events. Maskable Interrupts (on the INTR input) and Non-Maskable Interrupts (on the NMI input) are recognized at instruction boundaries. When NMI and maskable INTR are **both** recognized at the **same** instruction boundary, the Intel486 SX microprocessor/Intel OverDrive Processor invokes the NMI service routine first. If, after the NMI service routine has been invoked, maskable interrupts are still enabled, then the Intel486 SX microprocessor/Intel OverDrive Processor will invoke the appropriate interrupt service routine.

Table 2.18a. Intel486 SX Microprocessor/ Intel OverDrive Processor Priority for Invoking Service Routines in Case of Simultaneous External Interrupts

1. NMI
2. INTR

Exceptions are internally-generated events. Exceptions are detected by the Intel486 SX microprocessor/Intel OverDrive Processor if, in the course of executing an instruction, the Intel486 SX microprocessor/Intel OverDrive Processor detects a problematic condition. The Intel486 SX microprocessor/Intel OverDrive Processor then immediately invokes the appropriate exception service routine. The state of the Intel486 SX microprocessor/Intel OverDrive Processor is such that the instruction causing the exception can be restarted. If the exception service routine has taken care of the problematic condition, the instruction will execute without causing the same exception.

It is possible for a single instruction to generate several exceptions (for example, transferring a single operand could generate two page faults if the operand location spans two "not present" pages). However, only one exception is generated upon each attempt to execute the instruction. Each exception service routine should correct its corresponding exception, and restart the instruction. In this manner, exceptions are serviced until the instruction executes successfully.

As the Intel486 SX microprocessor/Intel OverDrive Processor executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2.18b. This cycle is repeated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

Table 2.18b. Sequence of Exception Checking

Consider the case of the Intel486 SX microprocessor/Intel OverDrive Processor having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed:

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
3. Check for external NMI and INTR.
4. Check for Segmentation Faults that prevented fetching the entire next instruction (exceptions 11 or 13).
5. Check for Page Faults that prevented fetching the entire next instruction (exception 14).
6. Check for Faults decoding the next instruction (exception 6 if illegal opcode; exception 6 if in Real Mode or in Virtual 8086 Mode and attempting to execute an instruction for Protected Mode only (see Section 4.6.4); or exception 13 if instruction is longer than 15 bytes, or privilege violation in Protected Mode (i.e., not at IOPL or at CPL=0).
7. If WAIT opcode, check if TS=1 and MP=1 (exception 7 if both are 1).
8. If opcode for Floating Point Unit, check if EM=1 or TS=1 (exception 7 if either are 1).
9. If opcode for Floating Point Unit (FPU), check FPU error status (exception 16 if error status is asserted).
10. Check in the following order for each memory reference required by the instruction:
    a. Check for Segmentation Faults that prevent transferring the entire memory quantity (exceptions 11, 12, 13).
    b. Check for Page Faults that prevent transferring the entire memory quantity (exception 14).

NOTE:
The order stated supports the concept of the paging mechanism being "underneath" the segmentation mechanism. Therefore, for any given code or data reference in memory, segmentation exceptions are generated before paging exceptions are generated.

2.7.7 INSTRUCTION RESTART

The Intel486 SX microprocessor/Intel OverDrive Processor fully supports restarting all instructions after faults. If an exception is detected in the instruction to be executed (exception categories 4 through 10 in Table 2.18b), the Intel486 SX microprocessor/Intel OverDrive Processor invokes the appropriate exception service routine. The Intel486 SX microprocessor/Intel OverDrive Processor is in a state that permits restart of the instruction, for all cases but those in Table 2.18c. Note that all such cases are easily avoided by proper design of the operating system.

Table 2.18c. Conditions Preventing Instruction Restart

An instruction causes a task switch to a task whose Task State Segment is **partially** "not present". (An entirely "not present" TSS is restartable.) Partially present TSS's can be avoided either by keeping the TSS's of such tasks present in memory, or by aligning TSS segments to reside entirely within a single 4K page (for TSS segments of 4 Kbytes or less).

NOTE:
These conditions are avoided by using the operating system designs mentioned in this table.

2.7.8 DOUBLE FAULT

A Double Fault (exception 8) results when the Intel486 SX microprocessor/Intel OverDrive Processor attempts to invoke an exception service routine for the segment exceptions (10, 11, 12 or 13), but in the process of doing so, detects an exception other than a Page Fault (exception 14).

A Double Fault (exception 8) will also be generated when the Intel486 SX microprocessor/Intel OverDrive Processor attempts to invoke the Page Fault (exception 14) service routine, and detects an exception other than a second Page Fault. In any functional system, the entire Page Fault service routine must remain "present" in memory.

When a Double Fault occurs, the Intel486 SX microprocessor/Intel OverDrive Processor invokes the exception service routine for exception 8.

2.7.9 FLOATING POINT INTERRUPT VECTORS

Several interrupt vectors of the Intel486 SX microprocessor/Intel OverDrive Processor are used to report exceptional conditions while executing numeric programs in either real or protected mode. Table 2.19 shows these interrupts and their causes.

Table 2.19. Interrupt Vectors Used by FPU

Interrupt Number	Cause of Interrupt
7	A Floating Point instruction was encountered when EM or TS of the Intel OverDrive Processor control register zero (CR0) was set. EM = 1 indicates that software emulation of the instruction is required. When TS is set, either a Floating Point or WAIT instruction causes interrupt 7. This indicates that the current FPU context may not belong to the current task.
13	The first word or doubleword of a numeric operand is not entirely within the limit of its segment. The return address pushed onto the stack of the exception handler points at the Floating Point instruction that caused the exception, including any prefixes. The FPU has not executed this instruction; the instruction pointer and data pointer register refer to a previous, correctly executed instruction.
16	The previous numerics instruction caused an unmasked exception. The address of the faulty instruction and the address of its operand are stored in the instruction pointer and data pointer registers. Only Floating Point and WAIT instructions can cause this interrupt. The Intel OverDrive Processor return address pushed onto the stack of the exception handler points to a WAIT or Floating Point instruction (including prefixes). This instruction can be restarted after clearing the exception condition in the FPU. The FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE instructions cannot cause this interrupt.

3.0 REAL MODE ARCHITECTURE

3.1 Real Mode Introduction

When the Intel486 SX microprocessor/Intel OverDrive Processor is reset or powered up, it is initialized in Real Mode. Real Mode has the same base architecture as the 8086, but allows access to the 32-bit register set of the Intel486 SX microprocessor/Intel OverDrive Processor. The addressing mechanism, memory size, interrupt handling, are all identical to the Real Mode on the 80286.

All of the Intel486 SX microprocessor/Intel OverDrive Processor instructions are available in Real Mode (except those instructions listed in Section 4.6.4). The default operand size in Real Mode is 16 bits, just like the 8086. In order to use the 32-bit registers and addressing modes, override prefixes must be used. In addition, the segment size on the Intel486 SX microprocessor/Intel OverDrive Processor in Real Mode is 64 Kbytes so 32-bit effective addresses must have a value less than 0000FFFFH. The primary purpose of Real Mode is to set up the processor for Protected Mode Operation.

The LOCK prefix on the Intel486 SX microprocessor/Intel OverDrive Processor even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the Intel486 SX microprocessor/Intel OverDrive Processor in Protected Mode and Virtual 8086 Mode. Paging makes it impossible to guarantee that repeated string instructions can be LOCKed. The Intel486 SX microprocessor/Intel OverDrive Processor can't require that all pages holding the string be physically present in memory. Hence, a Page Fault (exception 14) might have to be taken during the repeated string instruction. Therefore the LOCK prefix can't be supported during repeated string instructions.

These are the only instruction forms where the LOCK prefix is legal on the Intel486 SX microprocessor/Intel OverDrive Processor:

Opcode	Operands (Dest, Source)
BIT Test and SET/RESET/COMPLEMENT	Mem, Reg/immed
XCHG	Reg, Mem
XCHG	Mem, Reg
ADD, OR, ADC, SBB, AND, SUB, XOR	Mem, Reg/immed
NOT, NEG, INC, DEC	Mem
CMPXCHG, XADD	Mem, Reg

An exception 6 will be generated if a LOCK prefix is placed before any instruction form or opcode not listed above. The LOCK prefix allows indivisible read/modify/write operations on memory operands using the instructions above. For example, even the ADD Reg, Mem is not LOCKable, because the Mem operand is not the destination (and therefore no memory read/modify/operation is being performed).

Since, on the Intel486 SX microprocessor/Intel OverDrive Processor, repeated string instructions are not LOCKable, it is not possible to LOCK the bus for a long period of time. Therefore, the LOCK prefix is not IOPL-sensitive on the Intel486 SX microprocessor/Intel OverDrive Processor. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed above.

3.2 Memory Addressing

In Real Mode the maximum memory size is limited to 1 megabyte. Thus, only address lines A2–A19 are active. (Exception, after RESET address lines A20–A31 are high during CS-relative memory cycles until an intersegment jump or call is executed (see Section 6.5)).

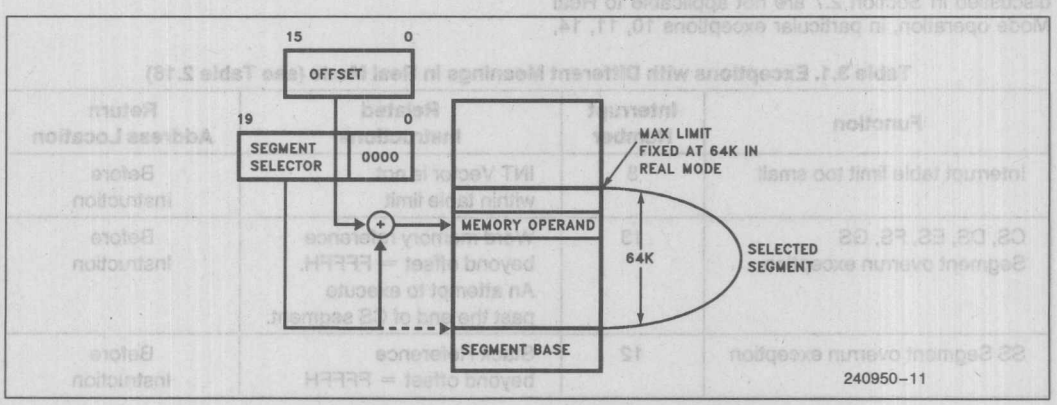


Figure 3.1. Real Address Mode Addressing

Since paging is not allowed in Real Mode the linear addresses are the same as physical addresses. Physical addresses are formed in Real Mode by adding the contents of the appropriate segment register which is shifted left by four bits to an effective address. This addition results in a physical address from 00000000H to 0010FFEFH. This is compatible with 80286 Real Mode. Since segment registers are shifted left by 4 bits, Real Mode segments always start on 16 byte boundaries.

All segments in Real Mode are exactly 64 Kbytes long, and may be read, written, or executed. The Intel486 SX microprocessor/Intel OverDrive Processor will generate an exception 13 if a data operand or instruction fetch occurs past the end of a segment (i.e., if an operand has an offset greater than FFFFH, for example a word with a low byte at FFFFH and the high byte at 0000H).

Segments may be overlapped in Real Mode. Thus, if a particular segment does not use all 64 Kbytes another segment can be overlayed on top of the unused portion of the previous segment. This allows the programmer to minimize the amount of physical memory needed for a program.

3.3 Reserved Locations

There are two fixed areas in memory which are reserved in Real address mode: system initialization area and the interrupt table area. Locations 00000H through 003FFH are reserved for interrupt vectors. Each one of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations FFFFFFF0H through FFFFFFFFH are reserved for system initialization.

3.4 Interrupts

Many of the exceptions shown in Table 2.17 and discussed in Section 2.7 are not applicable to Real Mode operation, in particular exceptions 10, 11, 14, 17, will not happen in Real Mode. Other exceptions have slightly different meanings in Real Mode; Table 3.1 identifies these exceptions.

3.5 Shutdown and Halt

The HLT instruction stops program execution and prevents the Intel486 SX microprocessor/Intel OverDrive Processor from using the local bus until restarted. Either NMI, INTR with interrupts enabled (IF = 1), or RESET will force the Intel486 SX microprocessor/Intel OverDrive Processor out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

As in the case in protected mode, the shutdown will occur when a severe error is detected that prevents further processing. In Real Mode, shutdown can occur under two conditions:

An interrupt or an exception occurs (exceptions 8 or 13) and the interrupt vector is larger than the Interrupt Descriptor Table (i.e., there is not an interrupt handler for the interrupt).

A CALL, INT or PUSH instruction attempts to wrap around the stack segment when SP is not even (i.e., pushing a value on the stack when SP = 0001 resulting in a stack segment greater than FFFFH).

An NMI input can bring the processor out of shutdown if the Interrupt Descriptor Table limit is large enough to contain the NMI interrupt vector (at least 0017H) and the stack has enough room to contain the vector and flag information (i.e., SP is greater than 0005H). If these conditions are not met, the Intel486 SX microprocessor/Intel OverDrive Processor is unable to execute the NMI and executes another shutdown cycle. In this case, the Intel486 SX microprocessor/Intel OverDrive Processor remains in the shutdown and can only exit via the RESET input.

Table 3.1. Exceptions with Different Meanings in Real Mode (see Table 2.16)

Function	Interrupt Number	Related Instructions	Return Address Location
Interrupt table limit too small	8	INT Vector is not within table limit	Before Instruction
CS, DS, ES, FS, GS Segment overrun exception	13	Word memory reference beyond offset = FFFFH. An attempt to execute past the end of CS segment.	Before Instruction
SS Segment overrun exception	12	Stack Reference beyond offset = FFFFH	Before Instruction

4.0 PROTECTED MODE ARCHITECTURE

4.1 Introduction

The complete capabilities of the Intel486 SX microprocessor/Intel OverDrive Processor are unlocked when the Intel486 SX microprocessor/Intel OverDrive Processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to four gigabytes (2^{32} bytes) and allows the running of virtual memory programs of almost unlimited size (64 terabytes or 2^{46} bytes). In addition Protected Mode allows the Intel486 SX microprocessor/Intel OverDrive Processor to run all of the existing 8086, 80286 and Intel386 microprocessor software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions especially optimized for supporting multitasking operating systems. The base architecture of the Intel486 SX microprocessor/Intel OverDrive Processor remains the same, the registers, instructions, and addressing modes described in the previous sections are retained. The main difference between Protected Mode, and Real Mode from a programmer's view is the increased address space, and a different addressing mechanism.

4.2 Addressing Mechanism

Like Real Mode, Protected Mode uses two components to form the logical address, a 16-bit selector is used to determine the linear base address of a segment, the base address is added to a 32-bit effective address to form a 32-bit linear address. The linear address is then either used as the 32-bit physical address, or if paging is enabled the paging mechanism maps the 32-bit linear address into a 32-bit physical address.

The difference between the two modes lies in calculating the base address. In Protected Mode the selector is used to specify an index into an operating system defined table (see Figure 4.1). The table contains the 32-bit base address of a given segment. The physical address is formed by adding the base address obtained from the table to the offset.

Paging provides an additional memory management mechanism which operates only in Protected Mode. Paging provides a means of managing the very large segments of the Intel486 SX microprocessor/Intel OverDrive Processor. As such, paging operates beneath segmentation. The paging mechanism translates the protected linear address which comes from the segmentation unit into a physical address. Figure 4.2 shows the complete Intel486 SX microprocessor/Intel OverDrive Processor addressing mechanism with paging enabled.

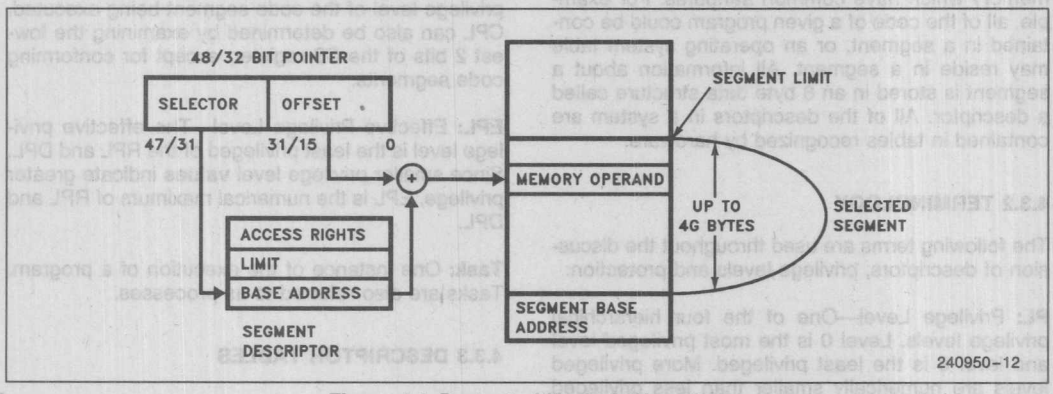


Figure 4.1. Protected Mode Addressing

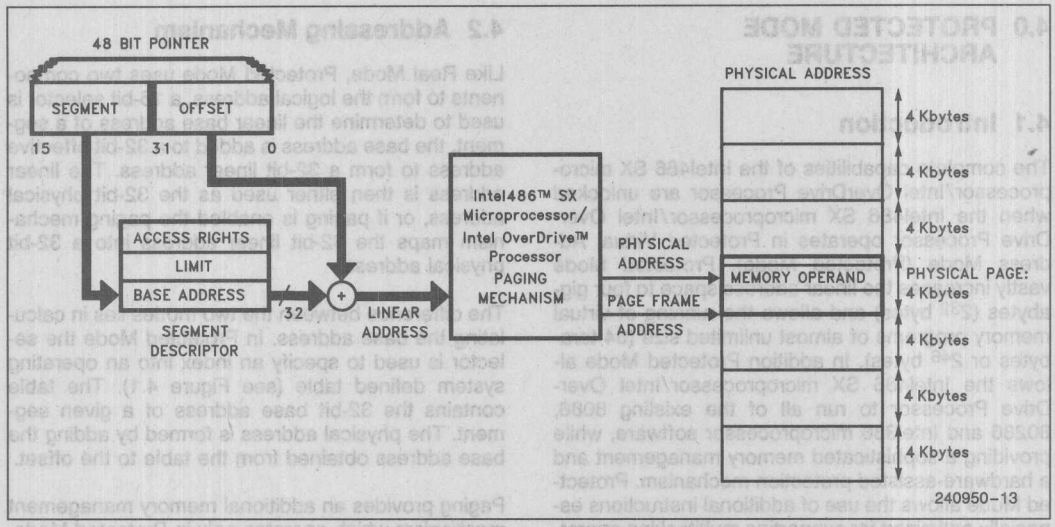


Figure 4.2. Paging and Segmentation

4.3 Segmentation

4.3.1 SEGMENTATION INTRODUCTION

Segmentation is one method of memory management. Segmentation provides the basis for protection. Segments are used to encapsulate regions of memory which have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about a segment is stored in an 8 byte data structure called a descriptor. All of the descriptors in a system are contained in tables recognized by hardware.

4.3.2 TERMINOLOGY

The following terms are used throughout the discussion of descriptors, privilege levels and protection:

PL: Privilege Level—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged. More privileged levels are numerically smaller than less privileged levels.

RPL: Requestor Privilege Level—The privilege level of the original supplier of the selector. RPL is determined by the **least two** significant bits of a selector.

DPL: Descriptor Privilege Level—This is the least privileged level at which a task may access that descriptor (and the segment associated with that descriptor). Descriptor Privilege Level is determined by bits 6:5 in the Access Right Byte of a descriptor.

CPL: Current Privilege Level—The privilege level at which a task is currently executing, which equals the privilege level of the code segment being executed. CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.

EPL: Effective Privilege Level—The effective privilege level is the least privileged of the RPL and DPL. Since smaller privilege level **values** indicate greater privilege, EPL is the numerical maximum of RPL and DPL.

Task: One instance of the execution of a program. Tasks are also referred to as processes.

4.3.3 DESCRIPTOR TABLES

4.3.3.1 Descriptor Tables Introduction

The descriptor tables define all of the segments which are used in an Intel486 SX microprocessor/Intel OverDrive Processor system. There are three

types of tables on the Intel486 SX microprocessor/ Intel OverDrive Processor which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays. They can range in size between 8 bytes and 64 Kbytes. Each table can hold up to 8192 8-byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address, and the 16-bit limit of each table.

Each of the tables has a register associated with it, the GDTR, LDTR, and the IDTR (see Figure 4.3). The LGDT, LLDT, and LIDT instructions, load the base and limit of the Global, Local, and Interrupt Descriptor Tables, respectively, into the appropriate register. The SGDT, SLDT, and SIDT store the base and limit values. These tables are manipulated by the operating system. Therefore, the load descriptor table instructions are privileged instructions.

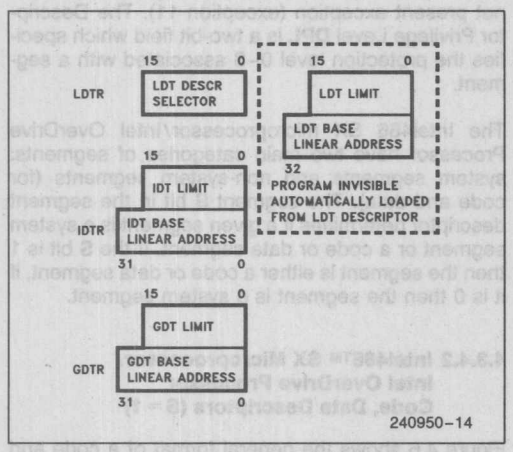


Figure 4.3. Descriptor Table Registers

4.3.3.2 Global Descriptor Table

The Global Descriptor Table (GDT) contains descriptors which are possibly available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for descriptors which are used for servicing interrupts (i.e., interrupt and trap descriptors). Every Intel486 SX microprocessor/Intel OverDrive Processor system contains a GDT. Generally the GDT contains code and data segments used by the operating systems and task state segments, and descriptors for the LDTs in a system.

The first slot of the Global Descriptor Table corresponds to the null selector and is not used. The null selector defines a null pointer value.

4.3.3.3 Local Descriptor Table

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks. A segment cannot be accessed by a task if its segment descriptor does not exist in either the current LDT or the GDT. This provides both isolation and protection for a task's segments, while still allowing global data to be shared among tasks.

Unlike the 6 byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT.

4.3.3.4 Interrupt Descriptor Table

The third table needed for Intel486 SX microprocessor/Intel OverDrive Processor systems is the Interrupt Descriptor Table. (See Figure 4.4.) The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced via INT instructions, external interrupt vectors, and exceptions. (See Section 2.7 **Interrupts).**

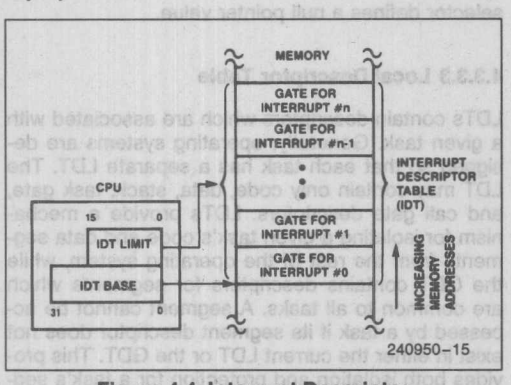


Figure 4.4. Interrupt Descriptor Table Register Use

4.3.4 DESCRIPTORS

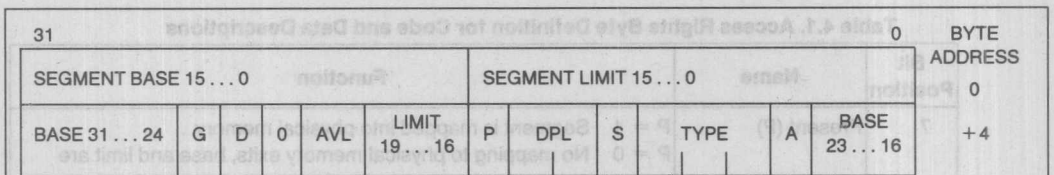
4.3.4.1 Descriptor Attribute Bits

The object to which the segment selector points to is called a descriptor. Descriptors are eight byte quantities which contain attributes about a given region of linear address space (i.e., a segment). These attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or 32-bit), and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 4.5 shows the general format of a descriptor. All segments on the Intel486 SX microprocessor/Intel OverDrive Processor have three attribute fields in common: the **P** bit, the **DPL** bit, and the **S** bit. The Present **P** bit is 1 if the segment is loaded in physical memory, if P = 0 then any attempt to access this segment causes a not present exception (exception 11). The Descriptor Privilege Level **DPL** is a two-bit field which specifies the protection level 0–3 associated with a segment.

The Intel486 SX microprocessor/Intel OverDrive Processor have two main categories of segments: system segments and non-system segments (for code and data). The segment **S** bit in the segment descriptor determines if a given segment is a system segment or a code or data segment. If the **S** bit is 1 then the segment is either a code or data segment, if it is 0 then the segment is a system segment.

4.3.4.2 Intel486™ SX Microprocessor/ Intel OverDrive Processor Code, Data Descriptors (S = 1)

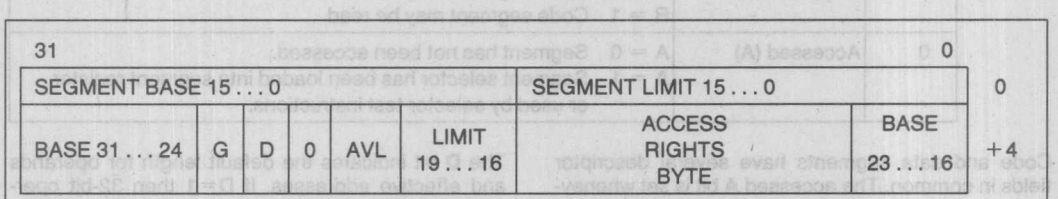
Figure 4.6 shows the general format of a code and data descriptor and Table 4.1 illustrates how the bits in the Access Rights Byte are interpreted.



BASE	Base Address of the segment
LIMIT	The length of the segment
P	Present Bit 1 = Present 0 = Not Present
DPL	Descriptor Privilege Level 0–3
S	Segment Descriptor 0 = System Descriptor 1 = Code or Data Segment Descriptor
TYPE	Type of Segment
A	Accessed Bit
G	Granularity Bit 1 = Segment length is page granular 0 = Segment length is byte granular
D	Default Operation Size (recognized in code segment descriptors only) 1 = 32-bit segment 0 = 16-bit segment
0	Bit must be zero (0) for compatibility with future processors
AVL	Available field for user or OS

NOTE:
In a maximum-size segment (i.e., a segment with G = 1 and segment limit 19...0 = FFFFFH), the lowest 12 bits of the segment base should be zero (i.e., segment base 11...000 = 000H).

Figure 4.5. Segment Descriptors



D/B	1 = Default Instruction Attributes are 32-Bits 0 = Default Instruction Attributes are 16-Bits
AVL	Available field for user or OS
G	Granularity Bit 1 = Segment length is page granular 0 = Segment length is byte granular
0	Bit must be zero (0) for compatibility with future processors

Figure 4.6. Segment Descriptors

Table 4.1. Access Rights Byte Definition for Code and Data Descriptions

	Bit Position	Name	Function	
	7	Present (P)	P = 1 Segment is mapped into physical memory. P = 0 No mapping to physical memory exits, base and limit are not used.	
	6–5	Descriptor Privilege Level (DPL)	Segment privilege attribute used in privilege tests.	
	4	Segment Descriptor (S)	S = 1 Code or Data (includes stacks) segment descriptor. S = 0 System Segment Descriptor or Gate Descriptor.	
Type Field Definition	3	Executable (E)	E = 0 Descriptor type is data segment:	If Data Segment (S = 1, E = 0)
	2	Expansion Direction (ED)	ED = 0 Expand up segment, offsets must be ≤ limit. ED = 1 Expand down segment, offsets must be > limit.	
	1	Writeable (W)	W = 0 Data segment may not be written into. W = 1 Data segment may be written into.	
	3	Executable (E)	E = 1 Descriptor type is code segment:	If Code Segment (S = 1, E = 1)
	2	Conforming (C)	C = 1 Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.	
	1	Readable (R)	R = 0 Code segment may not be read. R = 1 Code segment may be read.	
	0	Accessed (A)	A = 0 Segment has not been accessed. A = 1 Segment selector has been loaded into segment register or used by selector test instructions.	

Code and data segments have several descriptor fields in common. The accessed **A** bit is set whenever the processor accesses a descriptor. The **A** bit is used by operating systems to keep usage statistics on a given segment. The **G** bit, or granularity bit, specifies if a segment length is byte-granular or page-granular. Intel486 SX microprocessor/Intel OverDrive Processor segments can be one megabyte long with byte granularity (G=0) or four gigabytes with page granularity (G=1), (i.e., 2^{20} pages each page is 4 Kbytes in length). The granularity is totally unrelated to paging. An Intel486 SX microprocessor/Intel OverDrive Processor system can consist of segments with byte granularity, and page granularity, whether or not paging is enabled.

The executable **E** bit tells if a segment is a code or data segment. A code segment (E=1, S=1) may be execute-only or execute/read as determined by the Read **R** bit. Code segments are execute only if R=0, and execute/read if R=1. Code segments may never be written into.

NOTE:
Code segments may be modified via aliases. Aliases are writeable data segments which occupy the same range of linear address space as the code segment.

The **D** bit indicates the default length for operands and effective addresses. If D=1 then 32-bit operands and 32-bit addressing modes are assumed. If D=0 then 16-bit operands and 16-bit addressing modes are assumed. Therefore all existing 80286 code segments will execute on the Intel486 SX microprocessor/Intel OverDrive Processor assuming the D bit is set 0.

Another attribute of code segments is determined by the conforming **C** bit. Conforming segments, C=1, can be executed and shared by programs at different privilege levels. (See Section 4.4 **Protection**.)

Segments identified as data segments (E=0, S=1) are used for two types of Intel486 SX microprocessor/Intel OverDrive Processor segments: stack and data segments. The expansion direction **(ED)** bit specifies if a segment expands downward (stack) or upward (data). If a segment is a stack segment all offsets must be greater than the segment limit. On a data segment all offsets must be less than or equal to the limit. In other words, stack segments start at the base linear address plus the maximum segment limit and grow down to the base linear address plus the limit. On the other hand, data segments start at the base linear address and expand to the base linear address plus limit.

The write **W** bit controls the ability to write into a segment. Data segments are read-only if W = 0. The stack segment must have W = 1.

The **B** bit controls the size of the stack pointer register. If B = 1, then PUSHes, POPs, and CALLs all use the 32-bit ESP register for stack references and assume an upper limit of FFFFFFFFH. If B = 0, stack instructions all use the 16-bit SP register and assume an upper limit of FFFFH.

4.3.4.3 System Descriptor Formats

System segments describe information about operating system tables, tasks, and gates. Figure 4.7 shows the general format of system segment descriptors, and the various types of system segments. Intel486 SX microprocessor/Intel OverDrive Processor system descriptors contain a 32-bit base linear address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zero.

4.3.4.4 LDT Descriptors (S = 0, TYPE = 2)

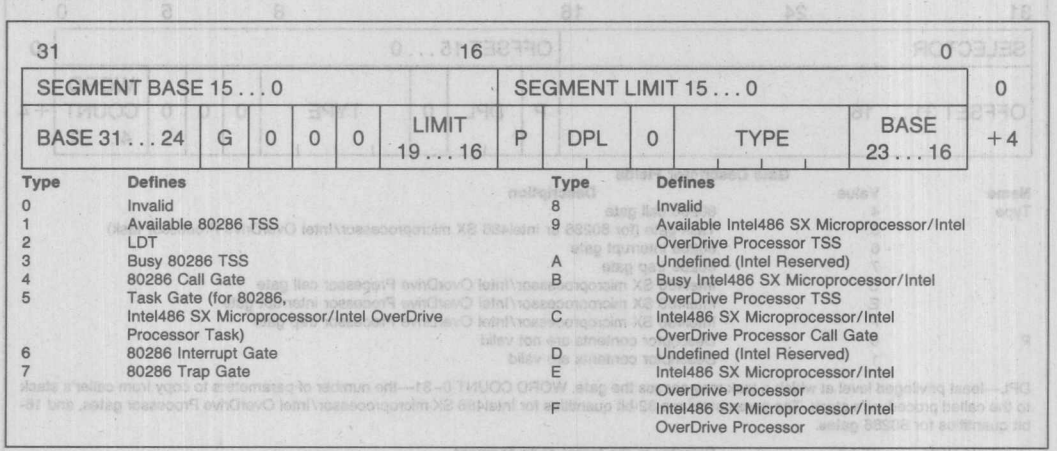
LDT descriptors (S = 0, TYPE = 2) contain information about Local Descriptor Tables. LDTs contain a table of segment descriptors, unique to a particular task. Since the instruction to load the LDTR is only available at privilege level 0, the DPL field is ignored. LDT descriptors are only allowed in the Global Descriptor Table (GDT).

4.3.4.5 TSS Descriptors (S = 0, TYPE = 1, 3, 9, B)

A Task State Segment (TSS) descriptor contains information about the location, size, and privilege level of a Task State Segment (TSS). A TSS in turn is a special fixed format segment which contains all the state information for a task and a linkage field to permit nesting tasks. The TYPE field is used to indicate whether the task is currently BUSY (i.e., on a chain of active tasks) or the TSS is available. The TYPE field also indicates if the segment contains a 80286, an Intel486 SX microprocessor TSS or an Intel OverDrive Processor TSS. The Task Register (TR) contains the selector which points to the current Task State Segment.

4.3.4.6 Gate Descriptors (S = 0, TYPE = 4–7, C, F)

Gates are used to control access to entry points within the target code segment. The various types of gate descriptors are **call** gates, **task** gates, **interrupt** gates, and **trap** gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the processor to automatically perform protection checks. It also allows system designers to control entry points to the operating system. Call gates are used to change privilege levels (see Section 4.4 **Protection**), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines.



Type	Defines
0	Invalid
1	Available 80286 TSS
2	LDT
3	Busy 80286 TSS
4	80286 Call Gate
5	Task Gate (for 80286, Intel486 SX Microprocessor/Intel OverDrive Processor Task)
6	80286 Interrupt Gate
7	80286 Trap Gate

Type	Defines
8	Invalid
9	Available Intel486 SX Microprocessor/Intel OverDrive Processor TSS
A	Undefined (Intel Reserved)
B	Busy Intel486 SX Microprocessor/Intel OverDrive Processor TSS
C	Intel486 SX Microprocessor/Intel OverDrive Processor Call Gate
D	Undefined (Intel Reserved)
E	Intel486 SX Microprocessor/Intel OverDrive Processor
F	Intel486 SX Microprocessor/Intel OverDrive Processor

Figure 4.7. System Segment Descriptors

Figure 4.8 shows the format of the four types of gate descriptors. Call gates are primarily used to transfer program control to a more privileged level. The call gate descriptor consists of three fields: the access byte, a long pointer (selector and offset) which points to the start of a routine and a word count which specifies how many parameters are to be copied from the caller's stack to the stack of the called routine. The word count field is only used by call gates when there is a change in the privilege level, other types of gates ignore the word count field.

Interrupt and trap gates use the destination selector and destination offset fields of the gate descriptor as a pointer to the start of the interrupt or trap handler routines. The difference between interrupt gates and trap gates is that the interrupt gate disables interrupts (resets the IF bit) while the trap gate does not.

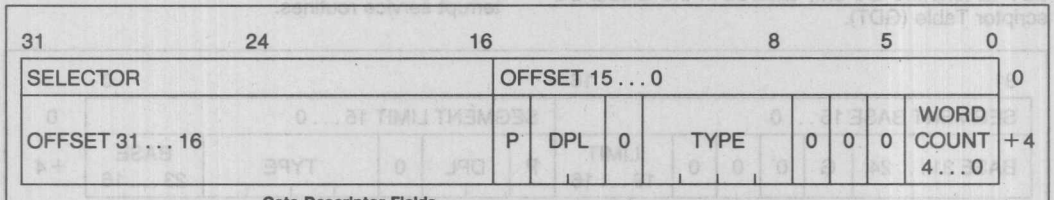
Task gates are used to switch tasks. Task gates may only refer to a task state segment (see Section 4.4.6 **Task Switching**) therefore only the destination selector portion of a task gate descriptor is used, and the destination offset is ignored.

Exception 13 is generated when a destination selector does not refer to a correct descriptor type, i.e., a code segment for an interrupt, trap or call gate, a TSS for a task gate.

The access byte format is the same for all gate descriptors. P=1 indicates that the gate contents are valid. P=0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (see Section 4.4 **Protection**). The S field, bit 4 of the access rights byte, must be 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 4.8.

4.3.4.7 Differences between Intel486™ SX Microprocessor/Intel OverDrive Processor and 80286 Descriptors

In order to provide operating system compatibility between the 80286 and Intel486 SX microprocessor/Intel OverDrive Processor, the Intel486 SX microprocessor/Intel OverDrive Processor supports all of the 80286 segment descriptors. Figure 4.9 shows the general format of an 80286 system segment descriptor. The only differences between 80286 and Intel486 SX microprocessor/Intel OverDrive Processor descriptor formats are that the values of the type fields, and the limit and base address fields have been expanded for the Intel486 SX microprocessor/Intel OverDrive Processor. The 80286 system segment descriptors contained a 24-bit base address and 16-bit limit, while the Intel486 SX microprocessor/Intel OverDrive Processor system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit.



Gate Descriptor Fields

Name	Value	Description
Type	4	80286 call gate
	5	Task gate (for 80286 or Intel486 SX microprocessor/Intel OverDrive Processor task)
	6	80286 interrupt gate
	7	80286 trap gate
	C	Intel486 SX microprocessor/Intel OverDrive Processor call gate
	E	Intel486 SX microprocessor/Intel OverDrive Processor interrupt gate
	F	Intel486 SX microprocessor/Intel OverDrive Processor trap gate
P	0	Descriptor contents are not valid
	1	Descriptor contents are valid

DPL—least privileged level at which a task may access the gate. WORD COUNT 0–31—the number of parameters to copy from caller's stack to the called procedure's stack. The parameters are 32-bit quantities for Intel486 SX microprocessor/Intel OverDrive Processor gates, and 16-bit quantities for 80286 gates.

DESTINATION SELECTOR	16-bit selector	Selector to the target code segment or Selector to the target task state segment for task gate
DESTINATION OFFSET	offset 16-bit 80286 32-bit Intel486 SX microprocessor/Intel OverDrive Processor	Entry point within the target code segment

Figure 4.8. Gate Descriptor Formats

By supporting 80286 system segments the Intel486 SX microprocessor/Intel OverDrive Processor is able to execute 80286 application programs on an Intel486 SX microprocessor/Intel OverDrive Processor operating system. This is possible because the Intel486 SX microprocessor/Intel OverDrive Processor automatically understands which descriptors are 80286-style descriptors and which descriptors are Intel486 SX microprocessor/Intel OverDrive Processor-style descriptors. In particular, if the upper word of a descriptor is zero, then that descriptor is a 80286-style descriptor.

The only other differences between 80286-style descriptors and Intel486 SX microprocessor/Intel OverDrive Processor descriptors is the interpretation of the word count field of call gates and the B bit. The word count field specifies the number of 16-bit quantities to copy for 80286 call gates and 32-bit quantities for Intel486 SX microprocessor/Intel OverDrive Processor call gates. The B bit controls the size of PUSHes when using a call gate; if B = 0 PUSHes are 16 bits, if B = 1 PUSHes are 32 bits.

4.3.4.8 Selector Fields

A selector in Protected Mode has three fields: Local or Global Descriptor Table Indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 4.10. The TI bits select one of two memory-based tables of descriptors (the Global Descriptor Table or the Local Descriptor Table). The Index selects one of 8K descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

4.3.4.9 Segment Descriptor Cache

In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with that selector is automatically loaded (cached) on the chip. Once loaded, all references to that segment use the cached descriptor information instead of reaccessing the descriptor. The contents of the descriptor cache are not visible to the programmer. Since descriptor caches only change when a segment register is changed, programs which modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.

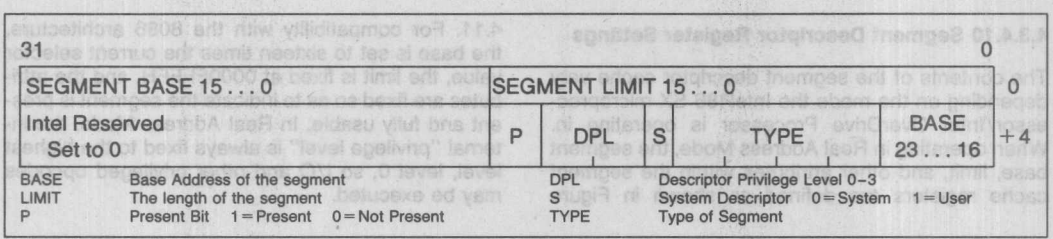


Figure 4.9. 80286 Code and Data Segment Descriptors

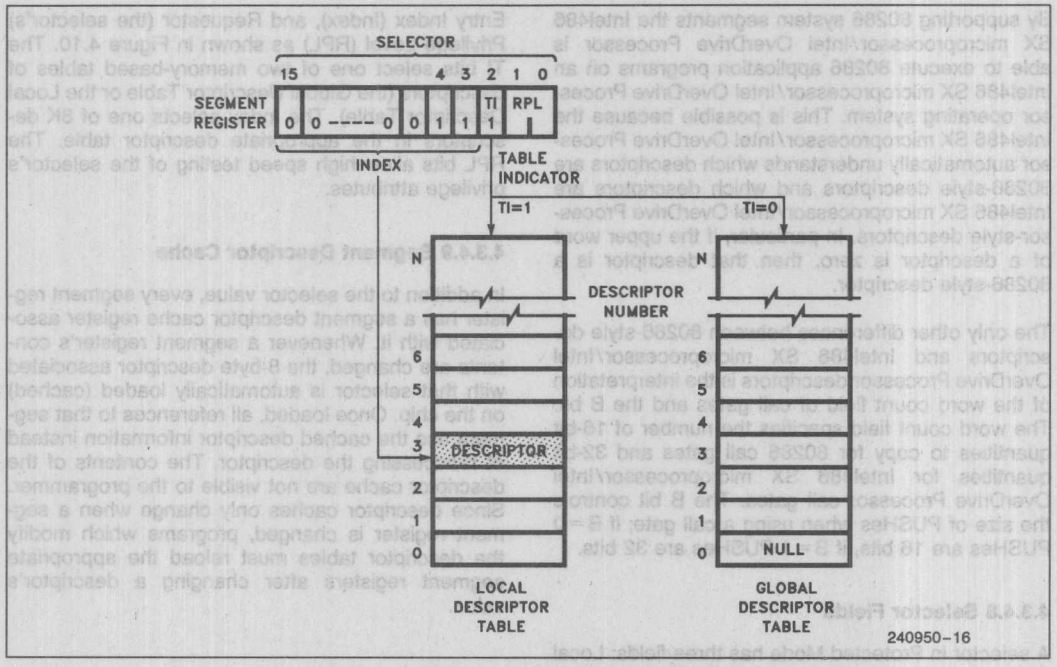
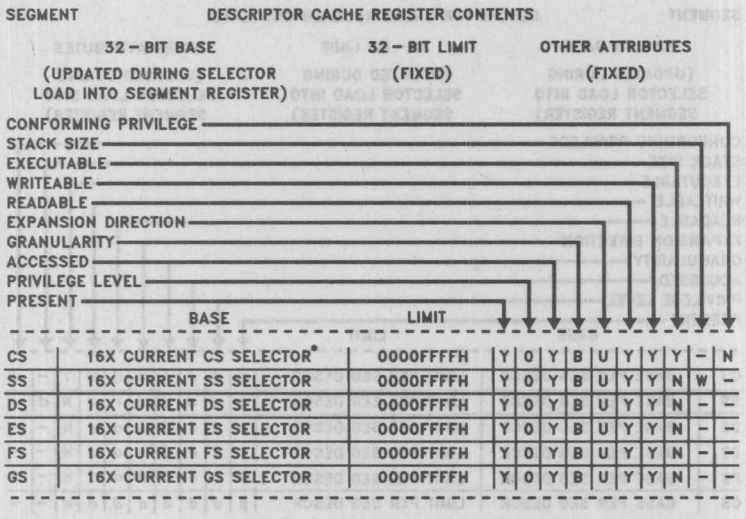


Figure 4.10. Example Descriptor Selection

4.3.4.10 Segment Descriptor Register Settings

The contents of the segment descriptor cache vary depending on the mode the Intel486 SX microprocessor/Intel OverDrive Processor is operating in. When operating in Real Address Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.11. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. In Real Address Mode, the internal "privilege level" is always fixed to the highest level, level 0, so I/O and other privileged opcodes may be executed.

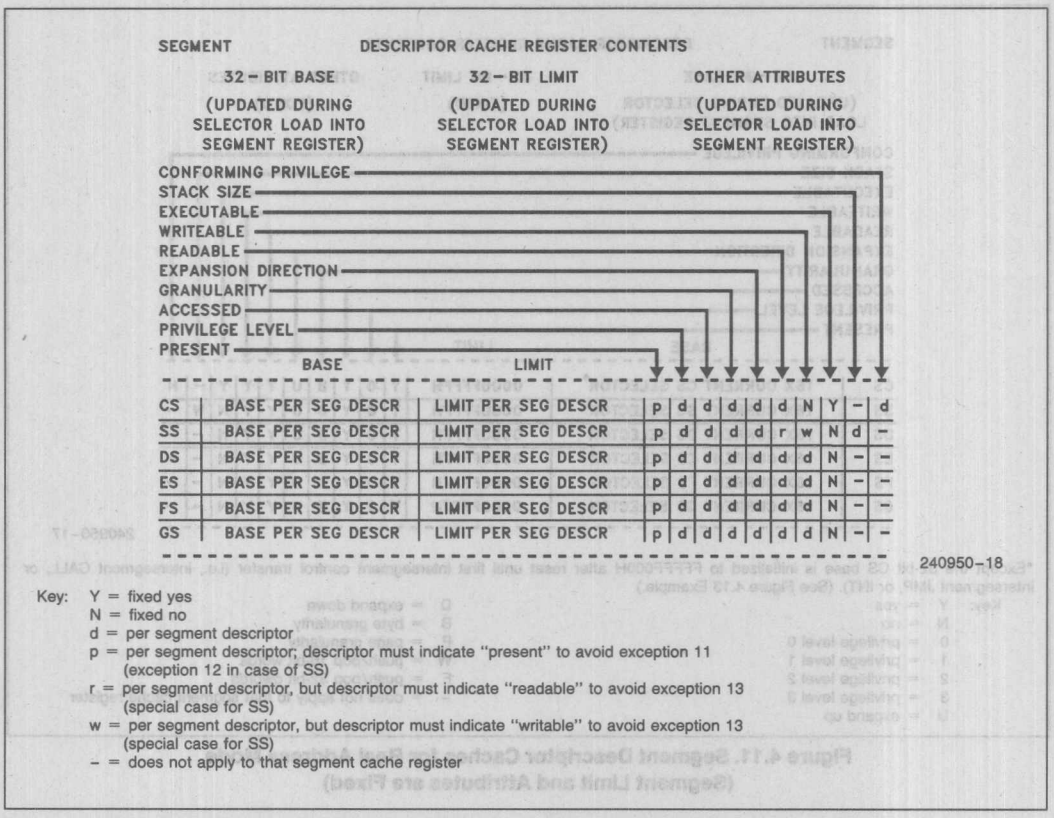


*Except the 32-bit CS base is initialized to FFFFF000H after reset until first intersegment control transfer (i.e., intersegment CALL, or intersegment JMP, or INT). (See Figure 4.13 Example.)

Key:
- Y = yes
- N = no
- 0 = privilege level 0
- 1 = privilege level 1
- 2 = privilege level 2
- 3 = privilege level 3
- U = expand up
- D = expand down
- B = byte granularity
- P = page granularity
- W = push/pop 16-bit words
- F = push/pop 32-bit dwords
- – = does not apply to that segment cache register

Figure 4.11. Segment Descriptor Caches for Real Address Mode (Segment Limit and Attributes are Fixed)

When operating in Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.12. In Protected Mode, each of these fields are defined according to the contents of the segment descriptor indexed by the selector value loaded into the segment register.



Key: Y = fixed yes
N = fixed no
d = per segment descriptor
p = per segment descriptor; descriptor must indicate "present" to avoid exception 11 (exception 12 in case of SS)
r = per segment descriptor, but descriptor must indicate "readable" to avoid exception 13 (special case for SS)
w = per segment descriptor, but descriptor must indicate "writable" to avoid exception 13 (special case for SS)
– = does not apply to that segment cache register

Figure 4.12. Segment Descriptor Caches for Protected Mode (Loaded per Descriptor)

When operating in a Virtual 8086 Mode within the Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.13. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. The virtual program executes at lowest privilege level, level 3, to allow trapping of all IOPL-sensitive instructions and level-0-only instructions.

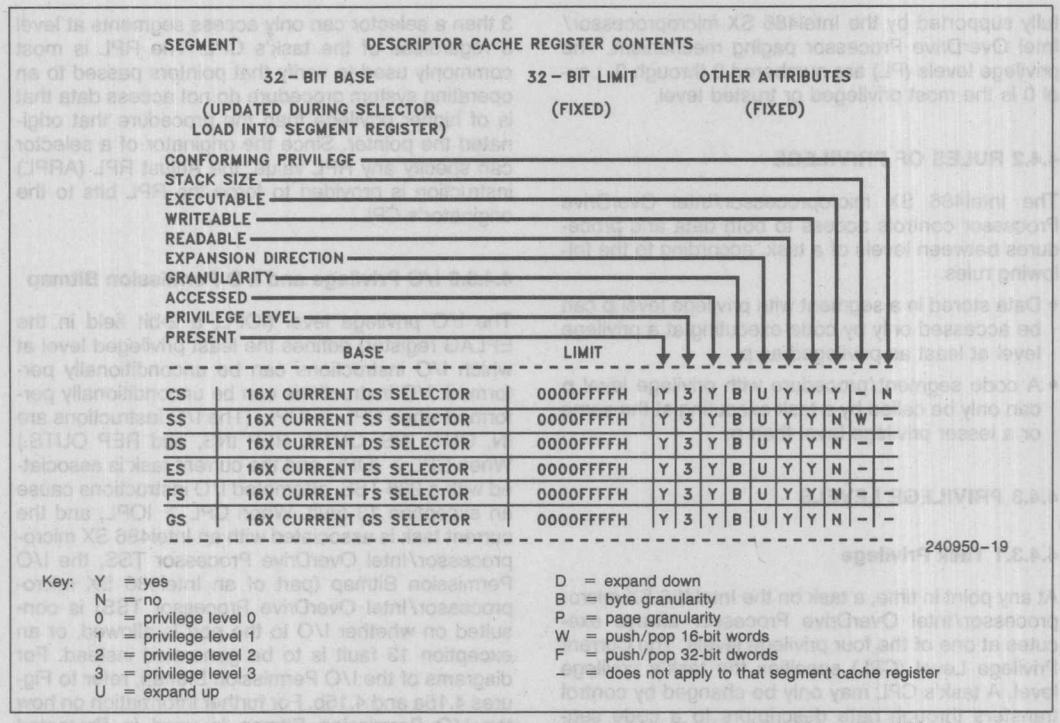


Figure 4.13. Segment Descriptor Caches for Virtual 8086 Mode within Protected Mode (Segment Limit and Attributes are Fixed)

4.4 Protection

4.4.1 PROTECTION CONCEPTS

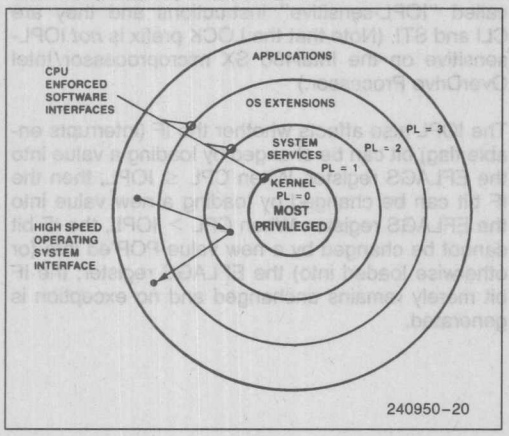


Figure 4.14. Four-Level Hierarchical Protection

The Intel486 SX microprocessor/Intel OverDrive Processor has four levels of protection which are optimized to support the needs of a multi-tasking operating system to isolate and protect user programs from each other and the operating system. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. Unlike traditional microprocessor-based systems where this protection is achieved only through the use of complex external hardware and software the Intel486 SX microprocessor/Intel OverDrive Processor provides the protection as part of its integrated Memory Management Unit. The Intel486 SX microprocessor/Intel OverDrive Processor offers an additional type of protection on a page basis, when paging is enabled (See Section 4.5.3 **Page Level Protection**).

The four-level hierarchical privilege system is illustrated in Figure 4-14. It is an extension of the user/supervisor privilege mode commonly used by minicomputers and, in fact, the user/supervisor mode is

fully supported by the Intel486 SX microprocessor/ Intel OverDrive Processor paging mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged or trusted level.

4.4.2 RULES OF PRIVILEGE

The Intel486 SX microprocessor/Intel OverDrive Processor controls access to both data and procedures between levels of a task, according to the following rules.

- Data stored in a segment with privilege level **p** can be accessed only by code executing at a privilege level at least as privileged as **p**.
- A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p**.

4.4.3 PRIVILEGE LEVELS

4.4.3.1 Task Privilege

At any point in time, a task on the Intel486 SX microprocessor/Intel OverDrive Processor always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies the task's privilege level. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level. (See Section 4.4.4 **Privilege Level Transfers**.) Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

4.4.3.2 Selector Privilege (RPL)

The privilege level of a selector is specified by the RPL field. The RPL is the two least significant bits of the selector. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level for the use of a segment. This level is called the task's effective privilege level (EPL). The EPL is defined as being the least privileged (i.e. numerically larger) level of a task's CPL and a selector's RPL. Thus, if selector's RPL = 0 then the CPL always specifies the privilege level for making an access using the selector. On the other hand if RPL = 3 then a selector can only access segments at level 3 regardless of the task's CPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Since the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

4.4.3.3 I/O Privilege and I/O Permission Bitmap

The I/O privilege level (IOPL, a 2-bit field in the EFLAG register) defines the least privileged level at which I/O instructions can be unconditionally performed. I/O instructions can be unconditionally performed when CPL ≤ IOPL. (The I/O instructions are IN, OUT, INS, OUTS, REP INS, and REP OUTS.) When CPL > IOPL, and the current task is associated with a 286 TSS, attempted I/O instructions cause an exception 13 fault. When CPL > IOPL, and the current task is associated with an Intel486 SX microprocessor/Intel OverDrive Processor TSS, the I/O Permission Bitmap (part of an Intel486 SX microprocessor/Intel OverDrive Processor TSS) is consulted on whether I/O to the port is allowed, or an exception 13 fault is to be generated instead. For diagrams of the I/O Permission Bitmap, refer to Figures 4.15a and 4.15b. For further information on how the I/O Permission Bitmap is used in Protected Mode or in Virtual 8086 Mode, refer to Section 4.6.4 Protection and I/O Permission Bitmap.

The I/O privilege level (IOPL) also affects whether several other instructions can be executed or cause an exception 13 fault instead. These instructions are called "IOPL-sensitive" instructions and they are CLI and STI. (Note that the LOCK prefix is *not* IOPL-sensitive on the Intel486 SX microprocessor/Intel OverDrive Processor.)

The IOPL also affects whether the IF (interrupts enable flag) bit can be changed by loading a value into the EFLAGS register. When CPL ≤ IOPL, then the IF bit can be changed by loading a new value into the EFLAGS register. When CPL > IOPL, the IF bit cannot be changed by a new value POP'ed into (or otherwise loaded into) the EFLAGS register; the IF bit merely remains unchanged and no exception is generated.

Table 4.2. Pointer Test Instructions

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

4.4.3.4 Privilege Validation

The Intel486 SX microprocessor/Intel OverDrive Processor provides several instructions to speed pointer testing and help maintain system integrity by verifying that the selector value refers to an appropriate segment. Table 4.2 summarizes the selector validation procedures available for the Intel486 SX microprocessor/Intel OverDrive Processor.

This pointer verification prevents the common problem of an application at PL = 3 calling a operating systems routine at PL = 0 and passing the operating system routine a "bad" pointer which corrupts a data structure belonging to the operating system. If the operating system routine uses the ARPL instruction to ensure that the RPL of the selector has no greater privilege than that of the caller, then this problem can be avoided.

4.4.3.5 Descriptor Access

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL as described above.

Any time an instruction loads data segment registers (DS, ES, FS, GS) the Intel486 SX microprocessor/Intel OverDrive Processor makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segments or readable code segments. The data access rules are specified in Section 4.4.2 **Rules of Privilege**. The only exception to those rules is readable conforming code segments which can be accessed at any privilege level.

Finally the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL an exception 13 (general protection fault) is generated.

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load selectors into SS must refer to data segment descriptors for writeable data segments. The DPL and RPL must equal the CPL. All other descriptor types or a privilege level violation will cause exception 13. A stack not present fault causes exception 12. Note that an exception 11 is used for a not-present code or data segment.

4.4.4 PRIVILEGE LEVEL TRANSFERS

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 4.3. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only via control transfers, by using gates, task switches, and interrupt or trap gates.

Table 4.3. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag register) = 0
**NT (Nested Task bit of flag register) = 1

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13 (e.g. JMP through a call gate, or IRET from a normal subroutine call).

In order to provide further system security, all control transfers are also subject to the privilege rules.

The privilege rules require that:

— Privilege level transitions can only occur via gates.

— JMPs can be made to a non-conforming code segment with the same privilege or to a conforming code segment with greater or equal privilege.

— CALLs can be made to a non-conforming code segment with the same privilege or via a gate to a more privileged level.

— Interrupts handled within the task obey the same privilege rules as CALLs.

— Conforming Code segments are accessible by privilege levels which are the same or less privileged than the conforming-code segment's DPL.

— Both the requested privilege level (RPL) in the selector pointing to the gate and the task's CPL must be of equal or greater privilege than the gate's DPL.

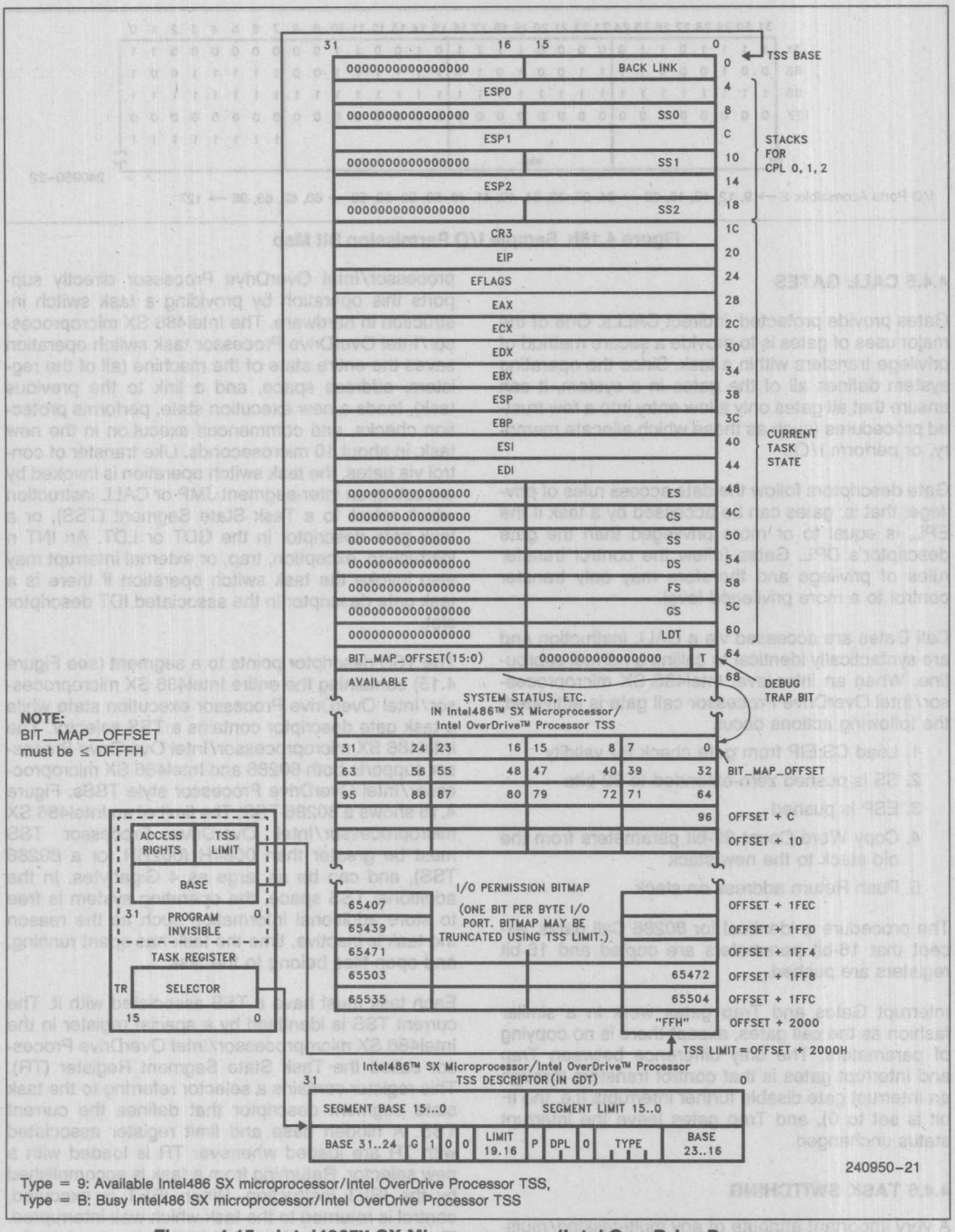
— The code segment selected in the gate must be the same or more privileged than the task's CPL.

— Return instructions that do not switch tasks can only return control to a code segment with same or less privilege.

— Task switches can be performed by a CALL, JMP, or INT which references either a task gate or task state segment who's DPL is less privileged or the same privilege as the old task's CPL.

Any control transfer that changes CPL within a task causes a change of stacks as a result of the privilege level change. The initial values of SS:ESP for privilege levels 0, 1, and 2 are retained in the task state segment (see Section 4.4.6 **Task Switching**). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and ESP registers and the previous stack pointer is pushed onto the new stack.

When RETurning to the original privilege level, use of the lower-privileged stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words (as specified in the gate's word count field) are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.



Type = 9: Available Intel486 SX microprocessor/Intel OverDrive Processor TSS,
Type = B: Busy Intel486 SX microprocessor/Intel OverDrive Processor TSS

Figure 4.15a. Intel486™ SX Microprocessor/Intel OverDrive Processor TSS and TSS Registers

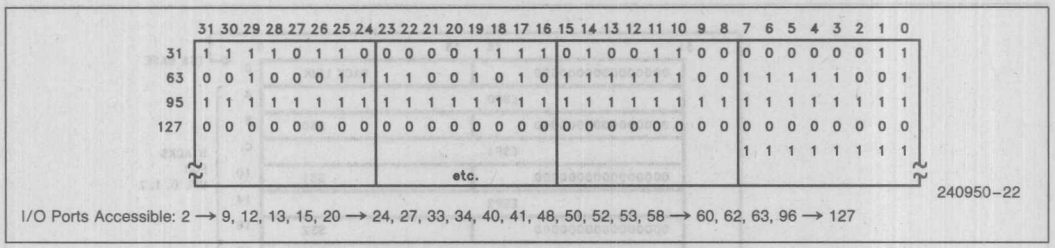


Figure 4.15b. Sample I/O Permission Bit Map

4.4.5 CALL GATES

Gates provide protected, indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures (such as those which allocate memory, or perform I/O).

Gate descriptors follow the data access rules of privilege; that is, gates can be accessed by a task if the EPL, is equal to or more privileged than the gate descriptor's DPL. Gates follow the control transfer rules of privilege and therefore may only transfer control to a more privileged level.

Call Gates are accessed via a CALL instruction and are syntactically identical to calling a normal subroutine. When an inter-level Intel486 SX microprocessor/Intel OverDrive Processor call gate is activated, the following actions occur.

1. Load CS:EIP from gate check for validity
2. SS is pushed zero-extended to 32 bits
3. ESP is pushed
4. Copy Word Count 32-bit parameters from the old stack to the new stack
5. Push Return address on stack

The procedure is identical for 80286 Call gates, except that 16-bit parameters are copied and 16-bit registers are pushed.

Interrupt Gates and Trap gates work in a similar fashion as the call gates, except there is no copying of parameters. The only difference between Trap and Interrupt gates is that control transfers through an Interrupt gate disable further interrupts (i.e. the IF bit is set to 0), and Trap gates leave the interrupt status unchanged.

4.4.6 TASK SWITCHING

A very important attribute of any multi-tasking/multi-user operating system is its ability to rapidly switch between tasks or processes. The Intel486 SX microprocessor/Intel OverDrive Processor directly supports this operation by providing a task switch instruction in hardware. The Intel486 SX microprocessor/Intel OverDrive Processor task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task, in about 10 microseconds. Like transfer of control via gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 4.15) containing the entire Intel486 SX microprocessor/Intel OverDrive Processor execution state while a task gate descriptor contains a TSS selector. The Intel486 SX microprocessor/Intel OverDrive Processor supports both 80286 and Intel486 SX microprocessor/Intel OverDrive Processor style TSSs. Figure 4.16 shows a 80286 TSS. The limit of an Intel486 SX microprocessor/Intel OverDrive Processor TSS must be greater than 0064H (002BH for a 80286 TSS), and can be as large as 4 Gigabytes. In the additional TSS space, the operating system is free to store additional information such as the reason the task is inactive, time the task has spent running, and open files belong to the task.

Each task must have a TSS associated with it. The current TSS is identified by a special register in the Intel486 SX microprocessor/Intel OverDrive Processor called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The current executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word (CR0) give information about the state of a task which are useful to the operating system. The Nested Task (NT) (bit 14 in EFLAGS) controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular return; when NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

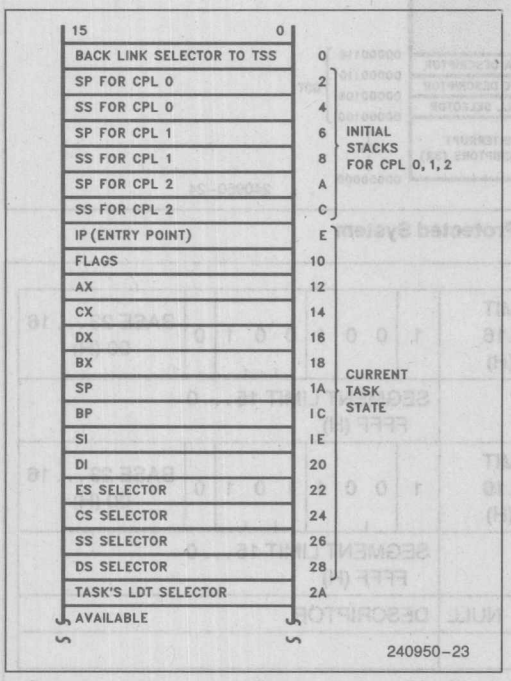


Figure 4.16. 80286 TSS

When a CALL or INT instruction initiates a task switch, the new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. (The NT bit will be restored after execution of the interrupt handler) NT may also be set or cleared by POPF or IRET instructions.

The Intel486 SX microprocessor/Intel OverDrive Processor task state segment is marked busy by changing the descriptor type field from TYPE 9H to TYPE BH. An 80286 TSS is marked busy by changing the descriptor type field from TYPE 1 to TYPE 3. Use of a selector that references a busy task state segment causes an exception 13.

The Virtual Mode (VM) bit 17 is used to indicate if a task, is a virtual 8086 task. If VM = 1, then the tasks will use the Real Mode addressing mechanism. The virtual 8086 environment is only entered and exited via a task switch (see Section 4.6 **Virtual Mode**).

The FPU's state is not automatically saved when a task switch occurs, because the incoming task may not use the FPU. The Task Switched (TS) Bit (bit 3 in the CR0) helps deal with the FPU's state in a multitasking environment. Whenever the Intel OverDrive Processor switches tasks, it sets the TS bit. The Intel OverDrive Processor detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the FPU. A processor extension not present exception (7) will occur when attempting to execute a Floating Point or WAIT instruction if the Task Switched and Monitor coprocessor extension bits are both set (i.e., TS = 1 and MP = 1).

The **T** bit in the Intel486 SX microprocessor/Intel OverDrive Processor TSS indicates that the processor should generate a debug exception when switching to a task. If T = 1 then upon entry to a new task a debug exception 1 will be generated.

4.4.7 INITIALIZATION AND TRANSITION TO PROTECTED MODE

Since the Intel486 SX microprocessor/Intel OverDrive Processor begins executing in Real Mode immediately after RESET it is necessary to initialize the system tables and registers with the appropriate values.

The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256 bytes long, and GDT must contain descriptors for the initial code, and data segments. Figure 4.17 shows the tables and Figure 4.18 the descriptors needed for a simple Protected Mode Intel486 SX microprocessor/Intel OverDrive Processor system. It has a single code and single data/stack segment each four gigabytes long and a single privilege level PL = 0.

The actual method of enabling Protected Mode is to load CR0 with the PE bit set, via the MOV CR0, R/M instruction. This puts the Intel486 SX microprocessor/Intel OverDrive Processor in Protected Mode.

After enabling Protected Mode, the next instruction should execute an intersegment JMP to load the CS register and flush the instruction decode queue. The final step is to load all of the data segment registers with the initial selector values.

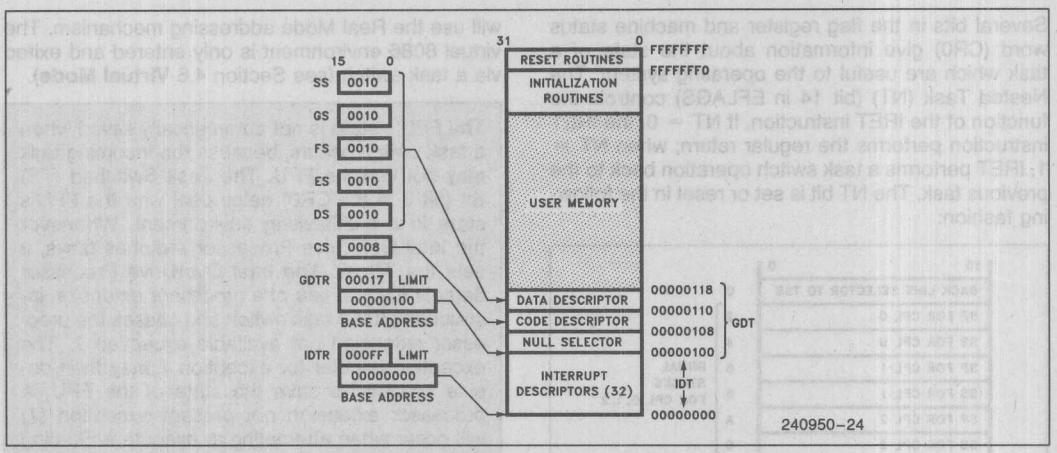


Figure 4.17. Simple Protected System

2	BASE 31 . . . 24 00 (H)	G 1	D 1	0	0	LIMIT 19.16 F (H)	1	0	0	1	0	0	1	0	BASE 23 . . . 16 00 (H)		
DATA DESCRIPTOR	SEGMENT BASE 15 . . . 0 0118 (H)						SEGMENT LIMIT 15 . . . 0 FFFF (H)										
1	BASE 31 . . . 24 00 (H)	G 1	D 1	0	0	LIMIT 19.16 F (H)	1	0	0	1	1	0	1	0	BASE 23 . . . 16 00 (H)		
CODE DESCRIPTOR	SEGMENT BASE 15 . . . 0 0118 (H)						SEGMENT LIMIT 15 . . . 0 FFFF (H)										
						NULL	DESCRIPTOR										
0																	
	31				24			16	15				8				0

Figure 4.18. GDT Descriptors for Simple System

An alternate approach to entering Protected Mode which is especially appropriate for multi-tasking operating systems, is to use the built in task-switch to load all of the registers. In this case the GDT would contain two TSS descriptors in addition to the code and data descriptors needed for the first task. The first JMP instruction in Protected Mode would jump to the TSS causing a task switch and loading all of the registers with the values stored in the TSS. The Task State Segment Register should be initialized to point to a valid TSS descriptor since a task switch saves the state of the current task in a task state segment.

4.4.8 TOOLS FOR BUILDING PROTECTED SYSTEMS

In order to simplify the design of a protected multi-tasking system, Intel provides a tool which allows the system designer an easy method of constructing the data structures needed for a Protected Mode Intel486 SX microprocessor/Intel OverDrive Processor system. This tool is the builder BLD-386. BLD-386 lets the operating system writer specify all of the segment descriptors discussed in the previous sections (LDTs, IDTs, GDTs, Gates, and TSSs) in a high-level language.

4.5 Paging

4.5.1 PAGING CONCEPTS

Paging is another type of memory management useful for virtual memory multitasking operating sys-

tems. Unlike segmentation which modularizes programs and data into variable length segments, paging divides programs into multiple uniform size pages. Pages bear no direct relation to the logical structure of a program. While segment selectors can be considered the logical "name" of a program module or data structure, a page most likely corresponds to only a portion of a module or data structure.

By taking advantage of the locality of reference displayed by most programs, only a small number of pages from each active task need be in memory at any one moment.

4.5.2 PAGING ORGANIZATION

4.5.2.1 Page Mechanism

The Intel486 SX microprocessor/Intel OverDrive Processor uses two levels of tables to translate the linear address (from the segmentation unit) into a physical address. There are three components to the paging mechanism of the Intel486 SX microprocessor/Intel OverDrive Processor: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the Intel486 SX microprocessor/Intel OverDrive Processor paging mechanism are the same size, namely, 4 Kbytes. A uniform size for all of the elements simplifies memory allocation and reallocation schemes, since there is no problem with memory fragmentation. Figure 4.19 shows how the paging mechanism works.

4.5.2.2 Page Descriptor Base Register

CR2 is the Page Fault Linear Address register. It holds the 32-bit linear address which caused the last page fault detected.

CR3 is the Page Directory Physical Base Address Register. It contains the physical starting address of the Page Directory. The lower 12 bits of CR3 are always zero to ensure that the Page Directory is always page aligned. Loading it via a MOV CR3, reg instruction causes the Page Table Entry cache to be flushed, as will a task switch through a TSS which **changes** the value of CR0. (See 4.5.5 **Translation Lookaside Buffer**.)

4.5.2.3 Page Directory

The Page Directory is 4 Kbytes long and allows up to 1024 Page Directory Entries. Each Page Directory Entry contains the address of the next level of tables, the Page Tables and information about the page table. The contents of a Page Directory Entry are shown in Figure 4.20. The upper 10 bits of the linear address (A22–A31) are used as an index to select the correct Page Directory Entry.

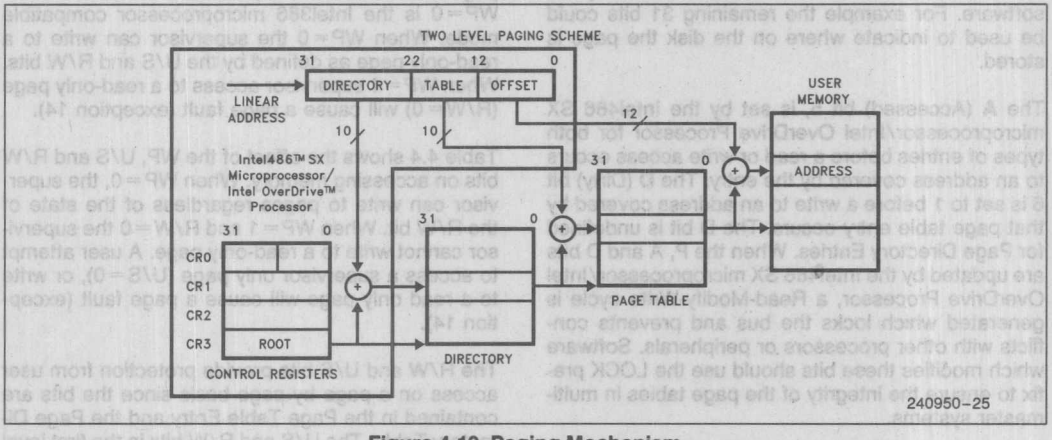


Figure 4.19. Paging Mechanism

31 12	11 10	9	8	7	6	5	4	3	2	1	0
PAGE TABLE ADDRESS 31..12	OS RESERVED		0	0	D	A	PCD	PWT	U/S	R/W	P

Figure 4.20. Page Directory Entry (Points to Page Table)

31 12	11 10	9	8	7	6	5	4	3	2	1	0
PAGE FRAME ADDRESS 31..12	OS RESERVED		0	0	D	A	PCD	PWT	U/S	R/W	P

Figure 4.21. Page Table Entry (Points to Page)

4.5.2.4 Page Tables

Each Page Table is 4 Kbytes and holds up to 1024 Page Table Entries. Page Table Entries contain the starting address of the page frame and statistical information about the page (see Figure 4.21). Address bits A12–A21 are used as an index to select one of the 1024 Page Table Entries. The 20 upper-bit page frame address is concatenated with the lower 12 bits of the linear address to form the physical address. Page tables can be shared between tasks and swapped to disks.

4.5.2.5 Page Directory/Table Entries

The lower 12 bits of the Page Table Entries and Page Directory Entries contain statistical information about pages and page tables respectively. The **P** (Present) bit 0 indicates if a Page Directory or Page Table entry can be used in address translation. If P = 1 the entry can be used for address translation, if P = 0 the entry cannot be used for translation, and all of the other bits are available for use by the software. For example the remaining 31 bits could be used to indicate where on the disk the page is stored.

The **A** (Accessed) bit 5, is set by the Intel486 SX microprocessor/Intel OverDrive Processor for both types of entries before a read or write access occurs to an address covered by the entry. The **D** (Dirty) bit 6 is set to 1 before a write to an address covered by that page table entry occurs. The D bit is undefined for Page Directory Entries. When the P, A and D bits are updated by the Intel486 SX microprocessor/Intel OverDrive Processor, a Read-Modify-Write cycle is generated which locks the bus and prevents conflicts with other processors or peripherals. Software which modifies these bits should use the LOCK prefix to ensure the integrity of the page tables in multi-master systems.

The 3 bits marked **OS Reserved** in Figure 4.20 and Figure 4.21 (bits 9–11) are software definable. OSs are free to use these bits for whatever purpose they wish. An example use of the **OS Reserved** bits would be to store information about page aging. By keeping track of how long a page has been in memory since being accessed, an operating system can implement a page replacement algorithm like Least Recently Used.

The (User/Supervisor) U/S bit 2 and the (Read/Write) R/W bit 1 are used to provide protection attributes for individual pages.

4.5.3 PAGE LEVEL PROTECTION (R/W, U/S BITS)

The Intel486 SX microprocessor/Intel OverDrive Processor provides a set of protection attributes for paging systems. The paging mechanism distinguishes between two levels of protection: User which corresponds to level 3 of the segmentation based protection, and supervisor which encompasses all of the other protection levels (0, 1, 2).

The R/W and U/S bits are used in conjunction with the WP bit in the flags register (EFLAGS). The Intel386 microprocessor does not contain the WP bit. The WP bit has been added to the Intel486 SX microprocessor/Intel OverDrive Processor to protect read-only pages from supervisor write accesses. The Intel386 microprocessor allows a read-only page to be written from protection levels 0, 1 or 2. WP=0 is the Intel386 microprocessor compatible mode. When WP=0 the supervisor can write to a read-only page as defined by the U/S and R/W bits. When WP=1 supervisor access to a read-only page (R/W=0) will cause a page fault (exception 14).

Table 4.4 shows the affect of the WP, U/S and R/W bits on accessing memory. When WP=0, the supervisor can write to pages regardless of the state of the R/W bit. When WP=1 and R/W=0 the supervisor cannot write to a read-only page. A user attempt to access a supervisor only page (U/S=0), or write to a read only page will cause a page fault (exception 14).

The R/W and U/S bits provide protection from user access on a page by page basis since the bits are contained in the Page Table Entry and the Page Directory Table. The U/S and R/W bits in the first level Page Directory Table apply to all entries in the page table pointed to by that directory entry. The U/S and R/W bits in the second level Page Table Entry apply only to the page described by that entry. The most restrictive of the U/S and R/W bits from the Page Directory Table and the Page Table Entry are used to address a page.

Example: If the U/S and R/W bits for the Page Directory entry were 10 (user read/execute) and the

U/S and R/W bits for the Page Table Entry were 01 (no user access at all), the access rights for the page would be 01, the numerically smaller of the two.

Note that a given segment can be easily made read-only for level 0, 1 or 2 via use of segmented protection mechanisms. (Section 4.4 **Protection.**)

4.5.4 PAGE CACHEABILITY (PWT AND PCD BITS)

PWT (page write through) and PCD (page cache disable) are two new bits defined in entries in both levels of the page table structure, the Page Directory Table and the Page Table Entry. PCD and PWT control page cacheability and write policy.

PWT controls write policy. PWT = 1 defines a write-through policy for the current page. PWT = 0 allows the possibility of write-back. PWT is ignored internally because the Intel486 SX microprocessor/Intel OverDrive Processor has a write-through cache. PWT can be used to control the write policy of a second level cache.

PCD controls cacheability. PCD = 0 enables caching in the on-chip cache. PCD alone does not enable caching, it must be conditioned by the KEN# (cache enable) input signal and the state of the CD (cache disable bit) and NW (no write-through) bits in control register 0 (CR0). When PCD = 1, caching is disabled regardless of the state of KEN#, CD and NW. (See Section 5.0, **On-Chip Cache.**)

The state of the PCD and PWT bits are driven out on the PCD and PWT pins during a memory access.

The PWT and PCD bits for a bus cycle are obtained either from control register 3 (CR3), the Page Directory Entry or the Page Table Entry, depending on the type of cycle run. However, when paging is disabled (PG = 0 in CR0) or for cycles which bypass paging (i.e., I/O (input/output) references, INTR (interrupt request) and HALT cycles), the PCD and PWT bits of CR3 are ignored. The Intel486 SX microprocessor/Intel OverDrive Processor assumes PCD = 0 and PWT = 0 and drives these values on the PCD and PWT pins.

When paging is enabled (PG = 1 in CR0), the bits from the page table entry are cached in the translation lookaside buffer (TLB), and are driven any time the page mapped by the TLB entry is referenced. For normal memory cycles run with paging enabled, the PWT and PCD bits are taken from the Page Table Entry. During TLB refresh cycles when the Page Directory and Page Table entries are read, the PWT and PCD bits must be obtained elsewhere. The bits are taken from CR3 when a Page Directory Entry is being read. The bits are taken from the Page Directory Entry when the Page Table Entry is being updated.

The PCD or PWT bits in CR3 are initialized to zero at reset, but can be set to any value by level 0 software.

4.5.5 TRANSLATION LOOKASIDE BUFFER

The Intel486 SX microprocessor/Intel OverDrive Processor paging hardware is designed to support demand paged virtual memory systems. However, performance would degrade substantially if the Intel486 SX microprocessor/Intel OverDrive Processor was required to access two levels of tables for every memory reference. To solve this problem, the Intel486 SX microprocessor/Intel OverDrive Processor keeps a cache of the most recently accessed pages. This cache is called the Translation Lookaside Buffer (TLB). The TLB is a four-way set associative 32-entry page table cache. It automatically keeps the most commonly used Page Table Entries in the Intel486 SX microprocessor/Intel OverDrive Processor. The 32-entry TLB coupled with a 4K page size, results in coverage of 128 Kbytes of memory addresses. For many common multi-tasking systems, the TLB will have a hit rate of about 98%. This means that the Intel486 SX microprocessor/Intel OverDrive Processor will only have to access the two-level page structure on 2% of all memory references. Figure 4.22 illustrates how the TLB complements the Intel486 SX microprocessor/Intel OverDrive Processor's paging mechanism.

Reading a new entry into the TLB (TLB refresh) is a two step process handled by the Intel486 SX microprocessor/Intel OverDrive Processor hardware. The sequence of data cycles to perform a TLB refresh are:

Table 4.4. Page Level Protection Attributes

U/S	R/W	WP	User Access	Supervisor Access
0	0	0	None	Read/Write/Execute
0	1	0	None	Read/Write/Execute
1	0	0	Read/Execute	Read/Write/Execute
1	1	0	Read/Write/Execute	Read/Write/Execute
0	0	1	None	Read/Execute
0	1	1	None	Read/Write/Execute
1	0	1	Read/Execute	Read/Execute
1	1	1	Read/Write/Execute	Read/Write/Execute

1. Read the correct Page Directory Entry, as pointed to by the page base register and the upper 10 bits of the linear address. The page base register is in control register 3.

1a. Optionally perform a locked read/write to set the accessed bit in the directory entry. The directory entry will actually get read twice if the Intel486 SX microprocessor/Intel OverDrive Processor needs to set any of the bits in the entry. If the page directory entry changes between the first and second reads, the data returned for the second read will be used.

2. Read the correct entry in the Page Table and place the entry in the TLB.

2a. Optionally perform a locked read/write to set the accessed and/or dirty bit in the page table entry. Again, note that the page table entry will actually get read twice if the Intel486 SX microprocessor/Intel OverDrive Processor needs to set any of the bits in the entry. Like the directory entry, if the data changes between the first and second read the data returned for the second read will be used.

Note that the directory entry must always be read into the Intel486 SX microprocessor/Intel OverDrive Processor, since directory entries are never placed in the paging TLB. Page faults can be signaled from either the page directory read or the page table read. Page directory and page table entries may be placed in the Intel486 SX microprocessor/Intel OverDrive Processor on-chip cache just like normal data.

4.5.6 PAGING OPERATION

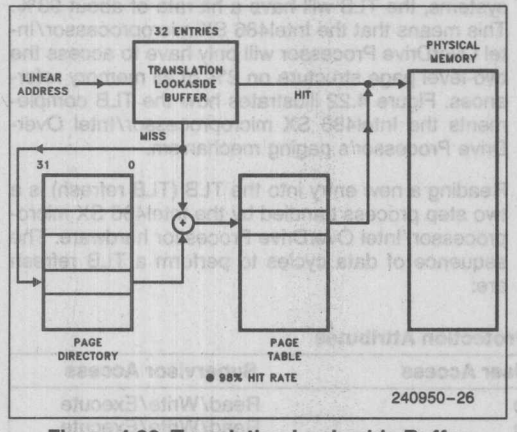


Figure 4.22. Translation Lookaside Buffer

The paging hardware operates in the following fashion. The paging unit hardware receives a 32-bit linear address from the segmentation unit. The upper 20 linear address bits are compared with all 32 entries in the TLB to determine if there is a match. If there is a match (i.e., a TLB hit), then the 32-bit physical address is calculated and will be placed on the address bus.

However, if the page table entry is not in the TLB, the Intel486 SX microprocessor/Intel OverDrive Processor will read the appropriate Page Directory Entry. If P = 1 on the Page Directory Entry indicating that the page table is in memory, then the Intel486 SX microprocessor/Intel OverDrive Processor will read the appropriate Page Table Entry and set the Access bit. If P = 1 on the Page Table Entry indicating that the page is in memory, the Intel486 SX microprocessor/Intel OverDrive Processor will update the Access and Dirty bits as needed and fetch the operand. The upper 20 bits of the linear address, read from the page table, will be stored in the TLB for future accesses. However, if P = 0 for either the Page Directory Entry or the Page Table Entry, then the Intel486 SX microprocessor/Intel OverDrive Processor will generate a page fault, an Exception 14.

The Intel486 SX microprocessor/Intel OverDrive Processor will also generate an exception 14 page fault if the memory reference violated the page protection attributes (i.e., U/S or R/W) (e.g., trying to write to a read-only page). CR2 will hold the linear address which caused the page fault. If a second page fault occurs, while the Intel486 SX microprocessor/Intel OverDrive Processor is attempting to enter the service routine for the first, then the Intel486 SX microprocessor/Intel OverDrive Processor will invoke the page fault (exception 14) handler a second time, rather than the double fault (exception 8) handler. Since Exception 14 is classified as a fault, CS: EIP will point to the instruction causing the page fault. The 16-bit error code pushed as part of the page fault handler will contain status bits which indicate the cause of the page fault.

The 16-bit error code is used by the operating system to determine how to handle the page fault. Figure 4.23a shows the format of the page-fault error code and the interpretation of the bits.

NOTE:
Even though the bits in the error code (U/S, W/R, and P) have similar names as the bits in the Page Directory/Table Entries, the interpretation of the error code bits is different. Figure 4.23b indicates what type of access caused the page fault.

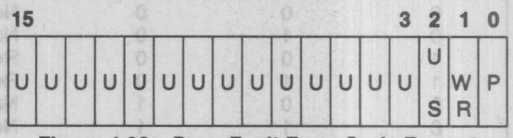


Figure 4.23a. Page Fault Error Code Format

U/S: The U/S bit indicates whether the access causing the fault occurred when the Intel486 SX microprocessor/Intel OverDrive Processor was executing in User Mode (U/S = 1) or in Supervisor mode (U/S = 0).

W/R: The W/R bit indicates whether the access causing the fault was a Read (W/R = 0) or a Write (W/R = 1).

P: The P bit indicates whether a page fault was caused by a not-present page (P = 0), or by a page level protection violation (P = 1).

U: UNDEFINED

U/S	W/R	Access Type
0	0	Supervisor* Read
0	1	Supervisor Write
1	0	User Read
1	1	User Write

*Descriptor table access will fault with U/S = 0, even if the program is executing at level 3.

Figure 4.23b. Type of Access Causing Page Fault

4.5.7 OPERATING SYSTEM RESPONSIBILITIES

The Intel486 SX microprocessor/Intel OverDrive Processor takes care of the page address translation process, relieving the burden from an operating system in a demand-paged system. The operating system is responsible for setting up the initial page tables, and handling any page faults. The operating system also is required to invalidate (i.e., flush) the TLB when any changes are made to any of the page table entries. The operating system must reload CR3 to cause the TLB to be flushed.

Setting up the tables is simply a matter of loading CR3 with the address of the Page Directory, and allocating space for the Page Directory and the Page Tables. The primary responsibility of the operating system is to implement a swapping policy and handle all of the page faults.

A final concern of the operating system is to ensure that the TLB cache matches the information in the paging tables. In particular, any time the operating system sets the P present bit of page table entry to zero, the TLB must be flushed. Operating systems may want to take advantage of the fact that CR3 is stored as part of a TSS, to give every task or group of tasks its own set of page tables.

4.6 Virtual 8086 Environment

4.6.1 EXECUTING 8086 PROGRAMS

The Intel486 SX microprocessor/Intel OverDrive Processor allows the execution of 8086 application programs in both Real Mode and in the Virtual 8086 Mode (Virtual Mode). Of the two methods, Virtual 8086 Mode offers the system designer the most flexibility. The Virtual 8086 Mode allows the execution of 8086 applications, while still allowing the system designer to take full advantage of the Intel486 SX microprocessor/Intel OverDrive Processor protection mechanism. In particular, the Intel486 SX microprocessor/Intel OverDrive Processor allows the simultaneous execution of 8086 operating systems and its applications, and an Intel486 SX microprocessor/Intel OverDrive Processor operating system and both 80286 and Intel486 SX microprocessor/Intel OverDrive Processor applications. Thus, in a multi-user Intel486 SX microprocessor/Intel OverDrive Processor computer, one person could be running an MS-DOS spreadsheet, another person using MS-DOS, and a third person could be running multiple Unix utilities and applications. Each person in this scenario would believe that he had the computer completely to himself. Figure 4.24 illustrates this concept.

4.6.2 VIRTUAL 8086 MODE ADDRESSING MECHANISM

One of the major differences between Intel486 SX microprocessor/Intel OverDrive Processor Real and Protected modes is how the segment selectors are interpreted. When the Intel486 SX microprocessor/Intel OverDrive Processor is executing in Virtual 8086 Mode the segment registers are used in an identical fashion to Real Mode. The contents of the segment register is shifted left 4 bits and added to the offset to form the segment base linear address.

The Intel486 SX microprocessor/Intel OverDrive Processor allows the operating system to specify which programs use the 8086 style address mechanism, and which programs use Protected Mode addressing, on a per task basis. Through the use of paging, the one megabyte address space of the Virtual Mode task can be mapped to anywhere in the 4 gigabyte linear address space of the Intel486 SX microprocessor/Intel OverDrive Processor. Like Real Mode, Virtual Mode effective addresses (i.e., segment offsets) that exceed 64 Kbyte will cause an exception 13. However, these restrictions should not prove to be important, because most tasks running in Virtual 8086 Mode will simply be existing 8086 application programs.

4.6.3 PAGING IN VIRTUAL MODE

The paging hardware allows the concurrent running of multiple Virtual Mode tasks, and provides protection and operating system isolation. Although it is not strictly necessary to have the paging hardware enabled to run Virtual Mode tasks, it is needed in order to run multiple Virtual Mode tasks or to relo-

cate the address space of a Virtual Mode task to physical address space greater than one megabyte.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into up to 256 pages. Each one of the pages can be located anywhere within the maximum 4 gigabyte physical address space of the Intel486 SX microprocessor/Intel OverDrive Processor. In addition, since CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations. Finally, the paging hardware allows the sharing of the 8086 operating system code between multiple 8086 applications. Figure 4.24 shows how the Intel486 SX microprocessor/Intel OverDrive Processor paging hardware enables multiple 8086 programs to run under a virtual memory demand paged system.

4.6.4 PROTECTION AND I/O PERMISSION BITMAP

All Virtual 8086 Mode programs execute at privilege level 3, the level of least privilege. As such, Virtual 8086 Mode programs are subject to all of the protection checks defined in Protected Mode. (This is different from Real Mode which implicitly is executing at privilege level 0, the level of greatest privilege.) Thus, an attempt to execute a privileged instruction when in Virtual 8086 Mode will cause an exception 13 fault.

The following are privileged instructions, which may be executed only at Privilege Level 0. Therefore, attempting to execute these instructions in Virtual 8086 Mode (or anytime CPL > 0) causes an exception 13 fault:

```
LIDT;  MOV DRn,reg;  MOV reg,DRn;
LGDT;  MOV TRn,reg;  MOV reg,TRn;
LMSW;  MOV CRn,reg;  MOV reg,CRn.
CLTS;
HLT;
```

Several instructions, particularly those applying to the multitasking model and protection model, are available only in Protected Mode. Therefore, attempting to execute the following instructions in Real Mode or in Virtual 8086 Mode generates an exception 6 fault:

```
LTR;   STR;
LLDT;  SLDT;
LAR;   VERR;
LSL;   VERW;
ARPL.
```

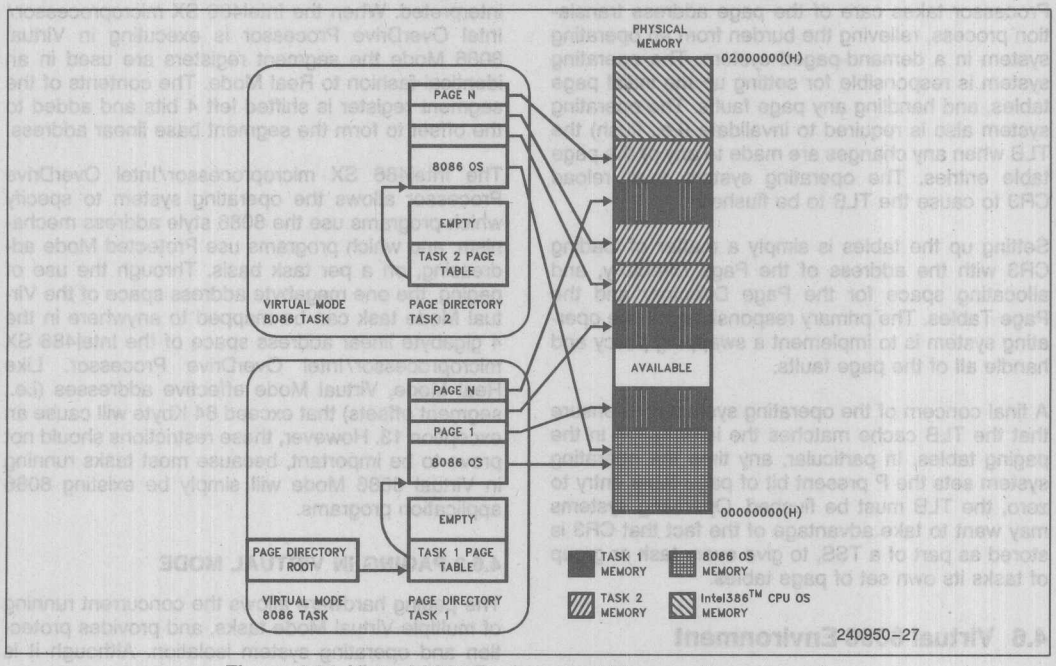


Figure 4.24. Virtual 8086 Environment Memory Management

The instructions which are IOPL-sensitive in Protected Mode are:

```
IN;          STI;
OUT;         CLI
INS;
OUTS;
REP INS;
REP OUTS;
```

In Virtual 8086 Mode, a slightly different set of instructions are made IOPL-sensitive. The following instructions are IOPL-sensitive in Virtual 8086 Mode:

```
INT n;       STI;
PUSHF;       CLI;
POPF;        IRET
```

The PUSHF, POPF, and IRET instructions are IOPL-sensitive in Virtual 8086 Mode only. This provision allows the IF flag (interrupt enable flag) to be virtualized to the Virtual 8086 Mode program. The INT n software interrupt instruction is also IOPL-sensitive in Virtual 8086 Mode. Note, however, that the INT 3 (opcode 0CCH), INTO, and BOUND instructions are not IOPL-sensitive in Virtual 8086 mode (they aren't IOPL sensitive in Protected Mode either).

Note that the I/O instructions (IN, OUT, INS, OUTS, REP INS, and REP OUTS) are **not** IOPL-sensitive in Virtual 8086 mode. Rather, the I/O instructions become automatically sensitive to the **I/O Permission Bitmap** contained in the **Intel486 SX microprocessor/Intel OverDrive Processor Task State Segment**. The I/O Permission Bitmap, automatically used by the Intel486 SX microprocessor/Intel OverDrive Processor in Virtual 8086 Mode, is illustrated by Figures 4.15a and 4.15b.

The I/O Permission Bitmap can be viewed as a 0–64 Kbit bit string, which begins in memory at offset Bit_Map_Offset in the current TSS. Bit_Map_Offset must be ≤ DFFFH so the entire bit map and the byte FFH which follows the bit map are all at offsets ≤ FFFFH from the TSS base. The 16-bit pointer Bit_Map_Offset (15:0) is found in the word beginning at offset 66H (102 decimal) from the TSS base, as shown in Figure 4.15a.

Each bit in the I/O Permission Bitmap corresponds to a single byte-wide I/O port, as illustrated in Figure 4.15a. If a bit is 0, I/O to the corresponding byte-wide port can occur without generating an exception. Otherwise the I/O instruction causes an exception 13 fault. Since every byte-wide I/O port must be protectable, all bits corresponding to a word-wide or dword-wide port must be 0 for the word-wide or dword-wide I/O to be permitted. If all the referenced bits are 0, the I/O will be allowed. If any referenced bits are 1, the attempted I/O will cause an exception 13 fault.

Due to the use of a pointer to the base of the I/O Permission Bitmap, the bitmap may be located anywhere within the TSS, or may be ignored completely by pointing the Bit_Map_Offset (15:0) beyond the limit of the TSS segment. In the same manner, only a small portion of the 64K I/O space need have an associated map bit, by adjusting the TSS limit to truncate the bitmap. This eliminates the commitment of 8K of memory when a complete bitmap is not required, while allowing the fully general case if desired.

EXAMPLE OF BITMAP FOR I/O PORTS 0–255: Setting the TSS limit to {bit_Map_Offset + 31 +1**} [** see note below] will allow a 32-byte bitmap for the I/O ports #0–255, plus a terminator byte of all 1's [** see note below]. This allows the I/O bitmap to control I/O Permission to I/O port 0–255 while causing an exception 13 fault on attempted I/O to any I/O port 80256 through 65,565.

****IMPORTANT IMPLEMENTATION NOTE:** Beyond the last byte of I/O mapping information in the I/O Permission Bitmap **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the Intel486 SX microprocessor/Intel OverDrive Processor TSS segment (see Figure 4.15a).

4.6.5 INTERRUPT HANDLING

In order to fully support the emulation of an 8086 machine, interrupts in Virtual 8086 Mode are handled in a unique fashion. When running in Virtual Mode all interrupts and exceptions involve a privilege change back to the host Intel486 SX microprocessor/Intel OverDrive Processor operating system. The Intel486 SX microprocessor/Intel OverDrive Processor operating system determines if the interrupt comes from a Protected Mode application or from a Virtual Mode program by examining the VM bit in the EFLAGS image stored on the stack.

When a Virtual Mode program is interrupted and execution passes to the interrupt routine at level 0, the VM bit is cleared. However, the VM bit is still set in the EFLAG image on the stack.

The Intel486 SX microprocessor/Intel OverDrive Processor operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The Intel486 SX microprocessor/Intel OverDrive Processor operating system may choose to let the 8086 operating system handle the interrupt or it may emulate the function of the interrupt handler. For example, many 8086 operating system calls are accessed by PUSHing parameters on the stack, and then executing an INT n instruction. If the IOPL is set to 0 then all INT n instructions will be intercepted by the Intel486 SX microproces-

sor/Intel OverDrive Processor operating system. The Intel486 SX microprocessor/Intel OverDrive Processor operating system could emulate the 8086 operating system's call. Figure 4.25 shows how the Intel486 SX microprocessor/Intel OverDrive Processor operating system could intercept an 8086 operating system's call to "Open a File".

An Intel486 SX microprocessor/Intel OverDrive Processor operating system can provide a Virtual 8086 Environment which is totally transparent to the application software via intercepting and then emulating 8086 operating system's calls, and intercepting IN and OUT instructions.

4.6.6 ENTERING AND LEAVING VIRTUAL 8086 MODE

Virtual 8086 mode is entered by executing an IRET instruction (at CPL = 0), or Task Switch (at any CPL) to an Intel486 SX microprocessor/Intel OverDrive Processor task whose Intel486 SX microprocessor/Intel OverDrive Processor TSS has a FLAGS image containing a 1 in the VM bit position while the Intel486 SX microprocessor/Intel OverDrive Processor is executing in Protected Mode. That is, one way to enter Virtual 8086 mode is to switch to a task with an Intel486 SX microprocessor/Intel OverDrive Processor TSS that has a 1 in the VM bit in the EFLAGS image. The other way is to execute a 32-bit IRET instruction at privilege level 0, where the stack has a 1 in the VM bit in the EFLAGS image. POPF does not affect the VM bit, even if the Intel486 SX microprocessor/Intel OverDrive Processor is in Protected Mode or level 0, and so cannot be used to enter Virtual 8086 Mode. PUSHF always pushes a 0 in the VM bit, even if the Intel486 SX microprocessor/Intel OverDrive Processor is in Virtual 8086 Mode, so that a program cannot tell if it is executing in REAL mode, or in Virtual 8086 mode.

The VM bit can be set by executing an IRET instruction only at privilege level 0, or by any instruction or Interrupt which causes a task switch in Protected Mode (with VM = 1 in the new FLAGS image), and can be cleared only by an interrupt or exception in Virtual 8086 Mode. IRET and POPF instructions executed in REAL mode or Virtual 8086 mode will not change the value in the VM bit.

The transition out of virtual 8086 mode to Intel486 SX microprocessor/Intel OverDrive Processor protected mode occurs only on receipt of an interrupt or exception (such as due to a sensitive instruction). In Virtual 8086 mode, all interrupts and exceptions vector through the protected mode IDT, and enter an interrupt handler in protected Intel486 SX microprocessor/Intel OverDrive Processor mode. That is, as part of interrupt processing, the VM bit is cleared.

Because the matching IRET must occur from level 0, if an Interrupt or Trap Gate is used to field an interrupt or exception out of Virtual 8086 mode, the Gate must perform an inter-level interrupt only to level 0. Interrupt or Trap Gates through conforming segments, or through segments with DPL > 0, will raise a GP fault with the CS selector as the error code.

4.6.6.1 Task Switches To/From Virtual 8086 Mode

Tasks which can execute in virtual 8086 mode must be described by a TSS with the new Intel486 SX microprocessor/Intel OverDrive Processor format (TYPE 9 or 11 descriptor).

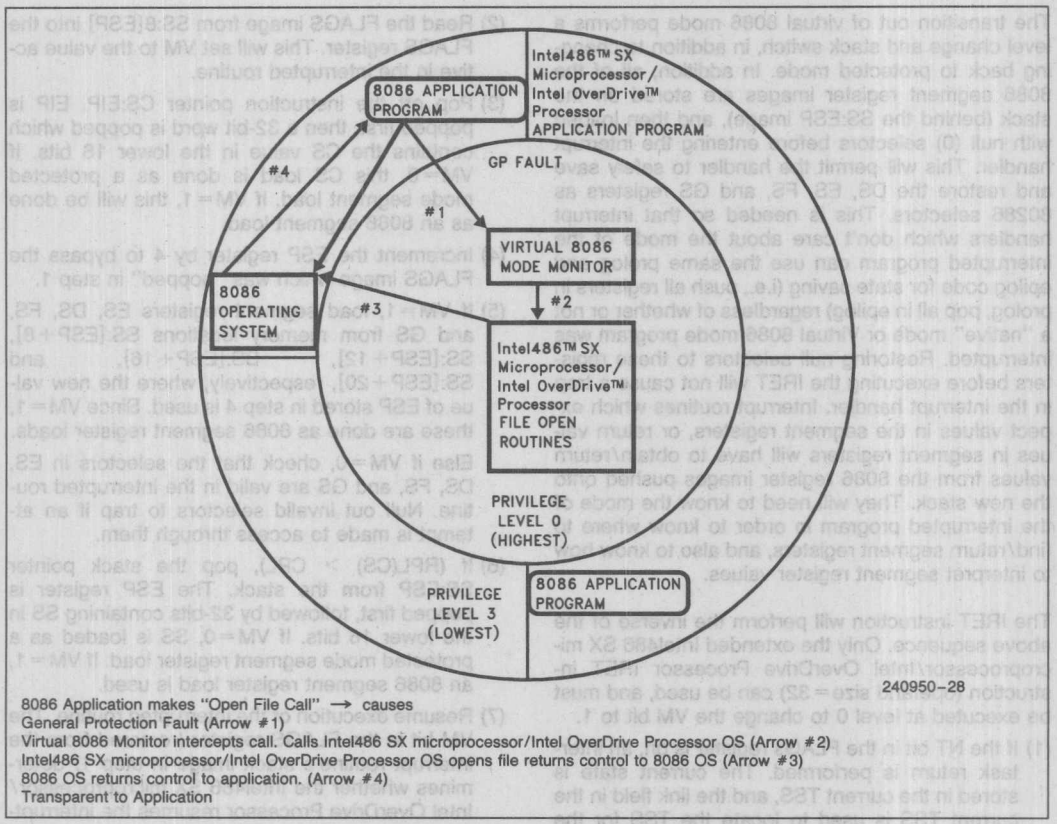
A task switch out of virtual 8086 mode will operate exactly the same as any other task switch out of a task with an Intel486 SX microprocessor/Intel OverDrive Processor TSS. All of the programmer visible state, including the FLAGS register with the VM bit set to 1, is stored in the TSS.

The segment registers in the TSS will contain 8086 segment base values rather than selectors.

A task switch into a task described by an Intel486 SX microprocessor/Intel OverDrive Processor TSS will have an additional check to determine if the incoming task should be resumed in virtual 8086 mode. Tasks described by 80286 format TSSs cannot be resumed in virtual 8086 mode, so no check is required there (the FLAGS image in 80286 format TSS has only the low order 16 FLAGS bits). Before loading the segment register images from an Intel486 SX microprocessor/Intel OverDrive Processor TSS, the FLAGS image is loaded, so that the segment registers are loaded from the TSS image as 8086 segment base values. The task is now ready to resume in virtual 8086 execution mode.

4.6.6.2 Transitions Through Trap and Interrupt Gates, and IRET

A task switch is one way to enter or exit virtual 8086 mode. The other method is to exit through a Trap or Interrupt gate, as part of handling an interrupt, and to enter as part of executing an IRET instruction. The transition out must use an Intel486 SX microprocessor/Intel OverDrive Processor Trap Gate (Type 14), or Intel486 SX microprocessor/Intel OverDrive Processor Interrupt Gate (Type 15), which must point to a non-conforming level 0 segment (DPL = 0) in order to permit the trap handler to IRET back to the Virtual 8086 program. The Gate must point to a non-conforming level 0 segment to perform a level switch to level 0 so that the matching IRET can change the VM bit. Intel486 SX microprocessor/Intel OverDrive Processor gates must be used, since 80286 gates save only the low 16 bits of



8086 Application makes "Open File Call" → causes
General Protection Fault (Arrow #1)
Virtual 8086 Monitor intercepts call. Calls Intel486 SX microprocessor/Intel OverDrive Processor OS (Arrow #2)
Intel486 SX microprocessor/Intel OverDrive Processor OS opens file returns control to 8086 OS (Arrow #3)
8086 OS returns control to application. (Arrow #4)
Transparent to Application

Figure 4.25. Virtual 8086 Environment Interrupt and Call Handling

the FLAGS register, so that the VM bit will not be saved on transitions through the 80286 gates. Also, the 16-bit IRET (presumably) used to terminate the 80286 interrupt handler will pop only the lower 16 bits from FLAGS, and will not affect the VM bit. The action taken for an Intel486 SX microprocessor/Intel OverDrive Processor Trap or Interrupt gate if an interrupt occurs while the task is executing in virtual 8086 mode is given by the following sequence.

(1) Save the FLAGS register in a temp to push later. Turn off the VM and TF bits, and if the interrupt is serviced by an Interrupt Gate, turn off IF also.

(2) Interrupt and Trap gates must perform a level switch from 3 (where the VM86 program executes) to level 0 (so IRET can return). This process involves a stack switch to the stack given in the TSS for privilege level 0. Save the Virtual 8086 Mode SS and ESP registers to push in a later step. The segment register load of SS will be done as a Protected Mode segment load, since the VM bit was turned off above.

(3) Push the 8086 segment register values onto the new stack, in the order: GS, FS, DS, ES. These are pushed as 32-bit quantities, with undefined values in the upper 16 bits. Then load these 4 registers with null selectors (0).

(4) Push the old 8086 stack pointer onto the new stack by pushing the SS register (as 32-bits, high bits undefined), then pushing the 32-bit ESP register saved above.

(5) Push the 32-bit FLAGS register saved in step 1.

(6) Push the old 8086 instruction pointer onto the new stack by pushing the CS register (as 32-bits, high bits undefined), then pushing the 32-bit EIP register.

(7) Load up the new CS:EIP value from the interrupt gate, and begin execution of the interrupt routine in protected Intel486 SX microprocessor/Intel OverDrive Processor mode.

The transition out of virtual 8086 mode performs a level change and stack switch, in addition to changing back to protected mode. In addition, all of the 8086 segment register images are stored on the stack (behind the SS:ESP image), and then loaded with null (0) selectors before entering the interrupt handler. This will permit the handler to safely save and restore the DS, ES, FS, and GS registers as 80286 selectors. This is needed so that interrupt handlers which don't care about the mode of the interrupted program can use the same prolog and epilog code for state saving (i.e., push all registers in prolog, pop all in epilog) regardless of whether or not a "native" mode or Virtual 8086 mode program was interrupted. Restoring null selectors to these registers before executing the IRET will not cause a trap in the interrupt handler. Interrupt routines which expect values in the segment registers, or return values in segment registers will have to obtain/return values from the 8086 register images pushed onto the new stack. They will need to know the mode of the interrupted program in order to know where to find/return segment registers, and also to know how to interpret segment register values.

The IRET instruction will perform the inverse of the above sequence. Only the extended Intel486 SX microprocessor/Intel OverDrive Processor IRET instruction (operand size = 32) can be used, and must be executed at level 0 to change the VM bit to 1.

(1) If the NT bit in the FLAGs register is on, an inter-task return is performed. The current state is stored in the current TSS, and the link field in the current TSS is used to locate the TSS for the interrupted task which is to be resumed.

    Otherwise, continue with the following sequence.

(2) Read the FLAGS image from SS:8[ESP] into the FLAGS register. This will set VM to the value active in the interrupted routine.

(3) Pop off the instruction pointer CS:EIP. EIP is popped first, then a 32-bit word is popped which contains the CS value in the lower 16 bits. If VM = 0, this CS load is done as a protected mode segment load. If VM = 1, this will be done as an 8086 segment load.

(4) Increment the ESP register by 4 to bypass the FLAGS image which was "popped" in step 1.

(5) If VM = 1, load segment registers ES, DS, FS, and GS from memory locations SS:[ESP + 8], SS:[ESP + 12], SS:[ESP + 16], and SS:[ESP + 20], respectively, where the new value of ESP stored in step 4 is used. Since VM = 1, these are done as 8086 segment register loads.

    Else if VM = 0, check that the selectors in ES, DS, FS, and GS are valid in the interrupted routine. Null out invalid selectors to trap if an attempt is made to access through them.

(6) If (RPL(CS) > CPL), pop the stack pointer SS:ESP from the stack. The ESP register is popped first, followed by 32-bits containing SS in the lower 16 bits. If VM = 0, SS is loaded as a protected mode segment register load. If VM = 1, an 8086 segment register load is used.

(7) Resume execution of the interrupted routine. The VM bit in the FLAGS register (restored from the interrupt routine's stack image in step 1) determines whether the Intel486 SX microprocessor/Intel OverDrive Processor resumes the interrupted routine in Protected mode of Virtual 8086 mode.

5.0 ON-CHIP CACHE

To meet its performance goals the Intel486 SX microprocessor/Intel OverDrive Processor contain an eight Kbyte cache. The cache is software transparent to maintain binary compatibility with previous generations of the Intel386™/Intel486™ family architecture.

The on-chip cache has been designed for maximum flexibility and performance. The cache has several operating modes offering flexibility during program execution and debugging. Memory areas can be defined as non-cacheable by software and external hardware. Protocols for cache line invalidations and replacement are implemented in hardware, easing system design.

5.1 Cache Organization

The on-chip cache is a unified code and data cache. The cache is used for both instruction and data accesses and acts on physical addresses.

The cache organization is 4-way set associative and each line is 16 bytes wide. The eight Kbytes of cache memory are logically organized as 128 sets, each containing four lines.

The cache memory is physically split into four 2-Kbyte blocks each containing 128 lines (see Figure 5.1). Associated with each 2-Kbyte block are 128 21-bit tags. There is a valid bit for each line in the cache. Each line in the cache is either valid or not valid. There are no provisions for partially valid lines.

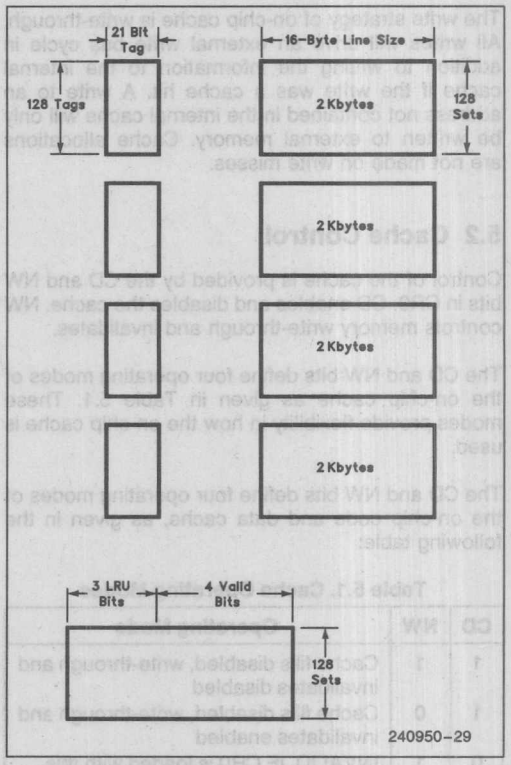


Figure 5.1. On-Chip Cache Physical Organization

The write strategy of on-chip cache is write-through. All writes will drive an external write bus cycle in addition to writing the information to the internal cache if the write was a cache hit. A write to an address not contained in the internal cache will only be written to external memory. Cache allocations are not made on write misses.

5.2 Cache Control

Control of the cache is provided by the CD and NW bits in CR0. CD enables and disables the cache. NW controls memory write-through and invalidates.

The CD and NW bits define four operating modes of the on-chip cache as given in Table 5.1. These modes provide flexibility in how the on-chip cache is used.

The CD and NW bits define four operating modes of the on-chip code and data cache, as given in the following table:

Table 5.1. Cache Operating Modes

CD	NW	Operating Mode
1	1	Cache fills disabled, write-through and invalidates disabled
1	0	Cache fills disabled, write-through and invalidates enabled
0	1	INVALID. IF CR0 is loaded with this configuration of bits, a GP fault with error code of 0 is raised.
0	0	Cache fills enabled, write-through and invalidates enabled

CD = 1, NW = 1

The cache is completely disabled by setting CD = 1 and NW = 1 and then flushing the cache. This mode may be useful for debugging programs where it is important to see all memory cycles at the pins. Writes which hit in the cache will not appear on the external bus.

It is possible to use the on-chip cache as fast static RAM by "pre-loading" certain memory areas into the cache and then setting CD = 1 and NW = 1. Pre-loading can be done by careful choice of memory references with the cache turned on or by use of the testability functions (see Section 8.2). When the cache is turned off the memory mapped by the cache is "frozen" into the cache since fills and invalidates are disabled.

CD = 1, NW = 0

Cache fills are disabled but write-throughs and invalidates are enabled. This mode is the same as if the KEN# pin was strapped HIGH disabling cache fills. Write-throughs and invalidates may still occur to keep the cache valid. This mode is useful if the software must disable the cache for a short period of time, and then re-enable it without flushing the original contents.

CD = 0, NW = 1

INVALID. If CR0 is loaded with this bit configuration, a General Protection fault with error code of 0 is raised. Note that this mode would imply a non-transparent write-back cache. A future processor may define this combination of bits to implement a write-back cache.

CD = 0, NW = 0

This is the normal operating mode.

Completely disabling the cache is a two step process. First CD and NW must be set to 1 and then the cache must be flushed. If the cache is not flushed, cache hits on reads will still occur and data will be read from the cache.

5.3 Cache Line Fills

Any area of memory can be cached in the Intel486 SX microprocessor/Intel OverDrive Processor. Non-cacheable portions of memory can be defined by the external system or by software. The external system can inform the Intel486 SX microprocessor/Intel OverDrive Processor that a memory address is non-cacheable by returning the KEN# pin inactive during a memory access (refer to Section 7.2.3). Software can prevent certain pages from being cached by setting the PCD bit in the page table entry.

A read request can be generated from program operation or by an instruction pre-fetch. The data will be supplied from the on-chip cache if a cache hit occurs on the read address. If the address is not in the cache, a read request for the data is generated on the external bus.

If the read request is to a cacheable portion of memory, the Intel486 SX microprocessor/Intel OverDrive Processor initiates a cache line fill. During a line fill a 16-byte line is read into the Intel486 SX microprocessor/Intel OverDrive Processor.

Cache fills will only be generated for read misses. Write misses will never cause a line in the internal cache to be allocated. If a cache hit occurs on a write, the line will be updated.

Cache line fills can be performed over 8- and 16-bit busses using the dynamic bus sizing feature. Refer to Section 7.1.3 for a description of dynamic bus sizing.

Refer to Section 7.2.3 for further information on cacheable cycles.

5.4 Cache Line Invalidations

The Intel486 SX microprocessor/Intel OverDrive Processor contain both a hardware and software mechanism for invalidating lines in its internal cache. Cache line invalidations are needed to keep the Intel486 SX microprocessor/Intel OverDrive Processor cache contents consistent with external memory.

Refer to Section 7.2.8 for further information on cache line invalidations.

5.5 Cache Replacement

When a line needs to be placed in its internal cache the Intel486 SX microprocessor/Intel OverDrive Processor first checks to see if there is a non-valid line in the set that can be replaced. If all four lines in the set are valid, a pseudo least-recently-used mechanism is used to determine which line should be replaced.

A valid bit is associated with each line in the cache. When a line needs to be placed in a set, the four valid bits are checked to see if there is a non-valid line that can be replaced. If a non-valid line is found, that line is marked for replacement.

The four lines in the set are labeled I0, I1, I2, and I3. The order in which the valid bits are checked during an invalidation is I0, I1, I2 and I3. All valid bits are cleared when the processor is reset or when the cache is flushed.

Replacement in the cache is handled by a pseudo least recently used (LRU) mechanism when all four lines in a set are valid. Three bits, B0, B1 and B2, are defined for each of the 128 sets in the cache. These bits are called the LRU bits. The LRU bits are updated for every hit or replace in the cache.

If the most recent access to the set was to I0 or I1, B0 is set to 1. B0 is set to 0 if the most recent access was to I2 or I3. If the most recent access to I0:I1 was to I0, B1 is set to 1, else B1 is set to 0. If the most recent access to I2:I3 was to I2, B2 is set to 1, else B2 is set to 0.

The pseudo LRU mechanism works in the following manner. When a line must be replaced, the cache will first select which of I0:I1 and I2:I3 was least recently used. Then the cache will determine which of the two lines was least recently used and mark it for replacement. This decision tree is shown in Figure 5.2. When the Intel486 SX microprocessor/Intel OverDrive Processor is reset or when the cache is flushed all 128 sets of three LRU bits are set to 0.

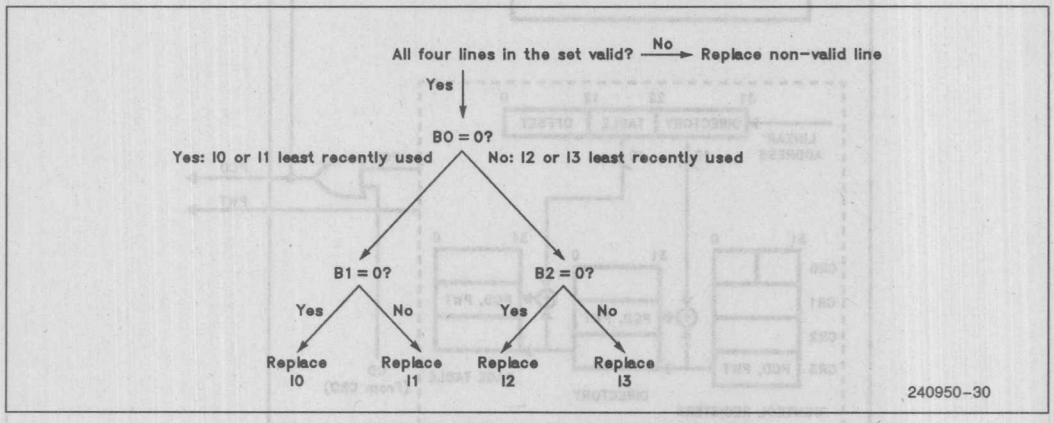


Figure 5.2. On-Chip Cache Replacement Strategy

5.6 Page Cacheability

Two bits for cache control, PWT and PCD, are defined in the page table and page directory entries. The state of these bits are driven out on the PWT and PCD pins during memory access cycles.

The PWT bit controls write policy for second level caches used with the Intel486 SX microprocessor/ Intel OverDrive Processor. Setting PWT = 1 defines a write-through policy for the current page while PWT = 0 allows the possibility of write-back. The state of PWT is ignored internally by the Intel486 SX microprocessor/Intel OverDrive Processor since the on-chip cache is write through.

The PCD bit controls cacheability on a page by page basis. The PCD bit is internally ANDed with the KEN# signal to control cacheability on a cycle by cycle basis (see Figure 5.3). PCD = 0 enables caching while PCD = 1 forbids it. Note that cache fills are enabled when PCD = 0 AND KEN# = 0. This logical AND is implemented physically with a NOR gate.

The state of the PCD bit in the page table entry is driven on the PCD pin when a page in external memory is accessed. The state of the PCD pin informs the external system of the cacheability of the requested information. The external system then returns KEN# telling the Intel486 SX microprocessor/ Intel OverDrive Processor if the area is cacheable. The Intel486 SX microprocessor/Intel OverDrive Processor initiates a cache line fill if PCD and KEN# indicate that the requested information is cacheable.

The PCD bit is masked with the CD (cache disable) bit in control register 0 to determine the state of the PCD pin. If CD = 1 the Intel486 SX microprocessor/ Intel OverDrive Processor forces the PCD pin HIGH. If CD = 0 the PCD pin is driven with the value for the page table entry/directory. See Figure 5.3.

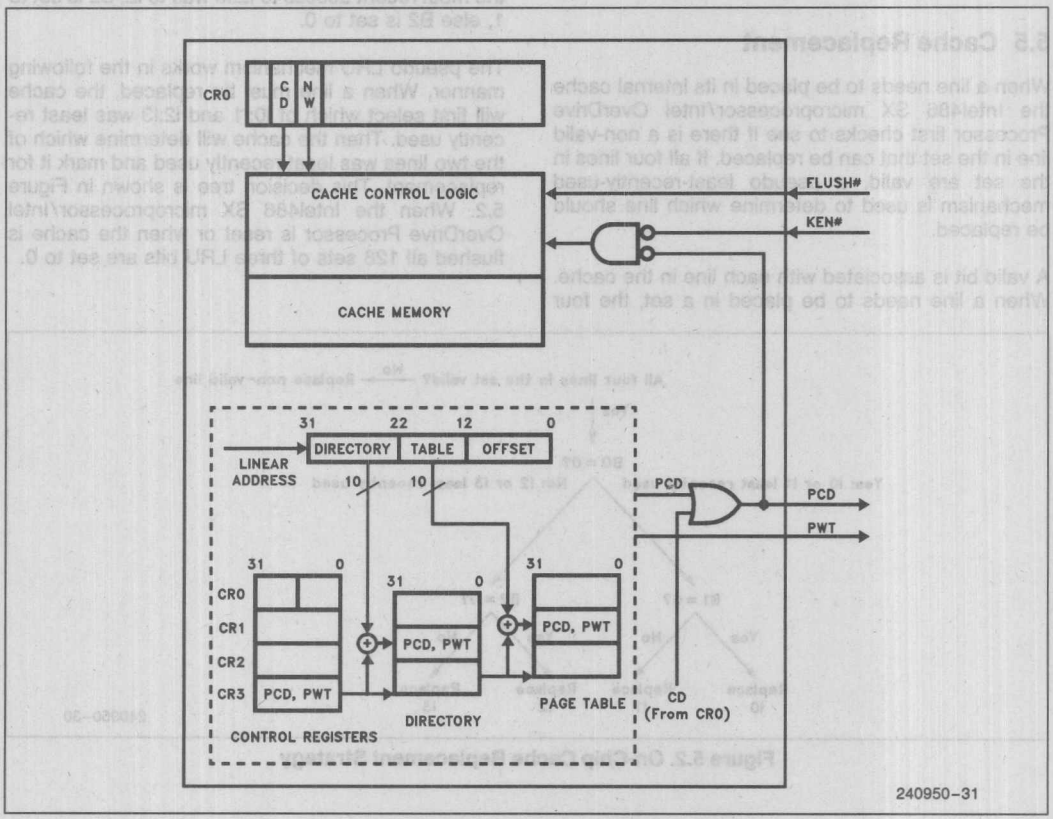


Figure 5.3. Page Cacheability

The PWT and PCD bits for a bus cycle are obtained from either CR3, the page directory or page table entry. These bits are assumed to be zero during real mode, whenever paging is disabled, or for cycles that bypass paging, (I/O references, interrupt acknowledge and Halt cycles).

When paging is enabled, the bits from the page table entry are cached in the TLB, and are driven any time the page mapped by the TLB entry is referenced. For normal memory cycles, PWT and PCD are taken from the page table entry. During TLB refresh cycles where the page table and directory entries are read, the PWT and PCD bits must be obtained elsewhere. During page table updates the bits are obtained from the page directory. When the page directory is updated the bits are obtained from CR3.

5.7 Cache Flushing

The on-chip cache can be flushed by external hardware or by software instructions. Flushing the cache clears all valid bits for all lines in the cache. The cache is flushed when external hardware asserts the FLUSH# pin.

The flush pin needs to be asserted for one clock if driven synchronously or for two clocks if driven asynchronously. The flush input is asynchronous but setup and hold times must be met. The flush pin should be deasserted after the cache flush is complete. Failure to deassert the pin will cause execution to stop as the processor will be repeatedly flushing the cache. If external hardware activates flush in response to an I/O write, flush must be asserted for at least two clocks prior to ready being returned for the I/O write. This ensures that the flush completes before the CPU begins execution of the instruction following the OUT instruction.

Flush is recognized during HOLD just like EADS#.

The instructions INVD and WBINVD cause the on-cache to be flushed. External caches connected to the Intel486 SX microprocessor/Intel OverDrive Processor are signaled to flush their contents when these instructions are executed.

WBINVD will cause an external write-back cache to write back dirty lines before flushing its contents. The external cache is signalled using the bus cycle definition pins and the byte enables (refer to Section 6.2.5 for the bus cycle definition pins and Section 7.2.11 for special bus cycles). Refer to the Intel486 microprocessor programmers reference manual for detailed instruction definitions.

The results of the INVD and WBINVD instructions are identical for the operation of the Intel486 SX microprocessor/Intel OverDrive Processor on-chip cache since the cache is write-through. Note that the INVD and WBINVD instructions are machine dependent. Future members of the Intel486 SX microprocessor/Intel OverDrive Processor family may change the definition of this instruction.

5.8 Caching Translation Lookaside Buffer Entries

The Intel486 SX microprocessor/Intel OverDrive Processor contain an integrated paging unit with a translation lookaside buffer (TLB). The TLB contains 32 entries. The TLB has been enhanced over the Intel386 microprocessor's TLB by upgrading the replacement strategy to a pseudo-LRU (least recently used) algorithm. The pseudo-LRU replacement algorithm is the same as that used in the on-chip cache.

The paging TLB operation is automatic whenever paging is enabled. The TLB contains the most recently used page table entries. A page table entry translates the linear address pointing to a particular page to the physical address where the page is stored in memory (refer to Section 4.5, **Paging**).

The paging unit will look up the linear address in the TLB in response to an internal bus request. The corresponding physical address is passed on to the on-chip cache or the external bus (in the event of a cache miss) when the linear address is present in the TLB.

The paging unit will access the page tables in external memory if the linear address is not in the TLB. The required page table entry will be read into the TLB and then the cache or bus cycle for the actual data will take place. The process of reading a new page table entry into the TLB is called a TLB refresh.

A TLB refresh is a two step process. The paging unit must first read the page directory entry which points to the appropriate page table. The page table entry to be stored in the TLB is then read from the page table. Control register 3 (CR3) points to the base of the page directory table.

The Intel486 SX microprocessor/Intel OverDrive Processor will allow page directory and page table entries (returned during TLB refreshes) to be stored in the on-chip cache. Setting the PCD bits in CR3 and the page directory entry to 1 will prevent the page directory and page table entries from being stored in the on-chip cache (see Section 5.6, **Page Cacheability**).

6.0 HARDWARE INTERFACE

6.1 Introduction

The Intel486 SX microprocessor/Intel OverDrive Processor bus has been designed to be similar to the Intel386 microprocessor bus whenever possible. Several new features have been added to the Intel486 SX microprocessor/Intel OverDrive Processor bus resulting in increased performance and functionality. New features include a 1X clock, a burst bus mechanism for high-speed internal cache fills, a cache line invalidation mechanism, enhanced bus arbitration capabilities, a BS8# bus sizing mechanism and parity support.

The Intel486 SX microprocessor/Intel OverDrive Processor are driven by a 1X clock as opposed to a 2X clock in the Intel386 microprocessor. A 1X clock allows simpler system design by cutting in half the clock speed required in the external system.

Like the Intel386 microprocessor, the Intel486 SX microprocessor/Intel OverDrive Processor have separate parallel busses for data and addresses. The bidirectional data bus is 32 bits in width. The address bus consists of two components: 30 address lines (A2–A31) and 4 byte enable lines (BE0#–BE3#). The address bus addresses external memory in the same manner as the Intel386 microprocessor: The address lines form the upper 30 bits of the address and the byte enables select individual bytes within a 4 byte location. The address lines are bidirectional for use in cache line invalidations.

The Intel486 SX microprocessor/Intel OverDrive Processor burst bus mechanism enables high-speed cache fills from external memory. Burst cycles can strobe data into the Intel486 SX microprocessor/Intel OverDrive Processor at a rate of one item every clock. Non-burst cycles have a maximum rate of one item every two clocks. Burst cycles are not limited to cache fills: all bus cycles requiring more than a single data cycle can be bursted.

The Intel486 SX microprocessor/Intel OverDrive Processor have a bus hold feature similar to that of the Intel386 microprocessor. During bus hold, the Intel486 SX microprocessor/Intel OverDrive Processor relinquish control of the local bus by floating its address, data and control busses.

The Intel486 SX microprocessor/Intel OverDrive Processor have an address hold feature in addition to bus hold. During address hold only the address bus is floated, the data and control busses can remain active. Address hold is used for cache line invalidations.

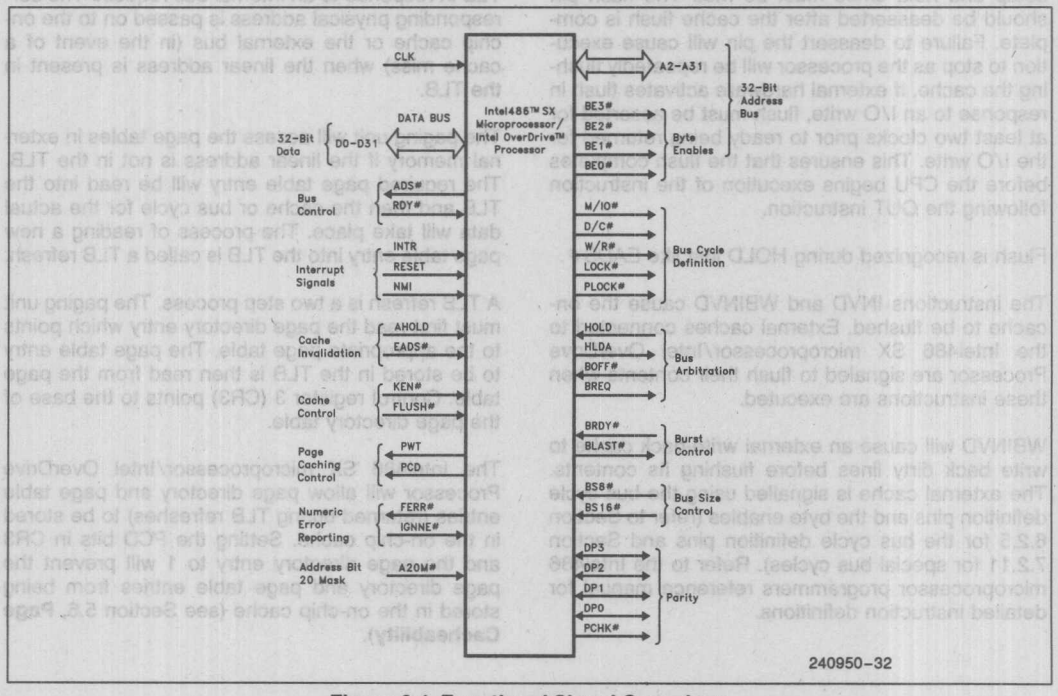


Figure 6.1. Functional Signal Groupings

Ahead is a brief description of the Intel486 SX microprocessor/Intel OverDrive Processor input and output signals arranged by functional groups. Before beginning the signal descriptions a few terms need to be defined. The # symbol at the end of a signal name indicates the active, or asserted, state occurs when the signal is at a low voltage. When a # is not present after the signal name, the signal is active at the high voltage level. The term "ready" is used to indicate that the cycle is terminated with RDY# or BRDY#.

Section 6 and 7 will discuss bus cycles and data cycles. A bus cycle is at least two clocks long and begins with ADS# active in the first clock and ready active in the last clock. Data is transferred to or from the Intel486 SX microprocessor/Intel OverDrive Processor during a data cycle. A bus cycle contains one or more data cycles.

6.2 Signal Descriptions

6.2.1 CLOCK (CLK)

CLK provides the fundamental timing and the internal operating frequency for the Intel486 SX microprocessor/Intel OverDrive Processor. All external timing parameters are specified with respect to the rising edge of CLK.

The Intel486 SX microprocessor/Intel OverDrive Processor can operate over a wide frequency range but CLK's frequency cannot change rapidly while RESET is inactive. CLK's frequency must be stable for proper chip operation since a single edge of CLK is used internally to generate two phases. CLK only needs TTL levels for proper operation. Figure 6.2 illustrates the CLK waveform.

6.2.2 Address Bus (A31–A2, BE0#–BE3#)

A31–A2 and BE0#–BE3# form the address bus and provide physical memory and I/O port addresses. The Intel486 SX microprocessor/Intel OverDrive Processor are capable of addressing 4 gigabytes of physical memory space (00000000H through FFFFFFFFH), and 64 Kbytes of I/O address space (00000000H through 0000FFFFH). A31–A2 identify addresses to a 4-byte location. BE0#–BE3# identify which bytes within the 4-byte location are involved in the current transfer.

Addresses are driven back into the Intel486 SX microprocessor/Intel OverDrive Processor over A31–A4 during cache line invalidations. The address lines are active HIGH. When used as inputs into the processor, A31–A4 must meet the setup and hold times, t_{22} and t_{23}. A31–A2 are not driven during bus or address hold.

The byte enable outputs, BE0#–BE3#, determine which bytes must be driven valid for read and write cycles to external memory.

BE3# applies to D24–D31

BE2# applies to D16–D23

BE1# applies to D8–D15

BE0# applies to D0–D7

BE0#–BE3# can be decoded to generate A0, A1 and BHE# signals used in 8- and 16-bit systems (see Table 7.5). BE0#–BE3# are active LOW and are not driven during bus hold.

6.2.3 DATA LINES (D31–D0)

The bidirectional lines, D31–D0, form the data bus for the Intel486 SX microprocessor/Intel OverDrive Processor D0–D7 define the least significant byte and D24–D31 the most significant byte.

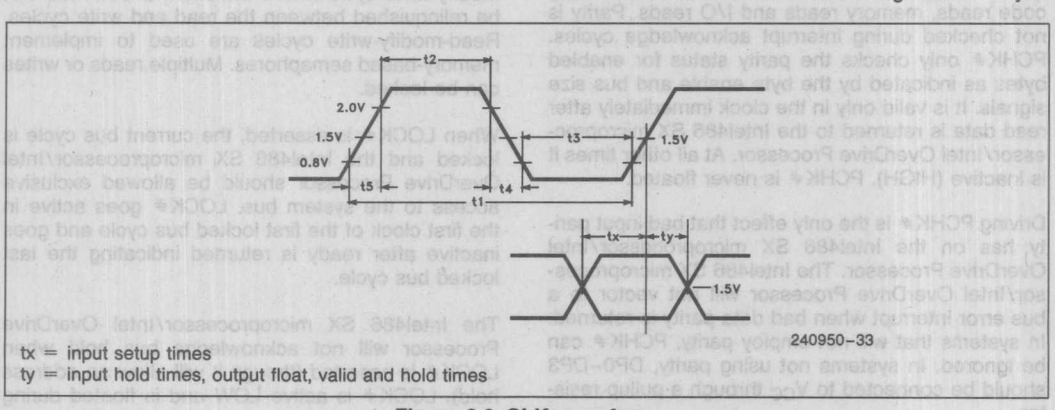


Figure 6.2. CLK waveform

Data transfers to 8- or 16-bit devices is possible using the data bus sizing feature controlled by the BS8# or BS16# input pins.

D31–D0 are active HIGH. For reads, D31–D0 must meet the setup and hold times, t_{22} and t_{23}. D31–D0 are not driven during read cycles and bus hold.

6.2.4 PARITY

Data Parity Input/Outputs (DP0–DP3)

DP0–DP3 are the data parity pins for the processor. There is one pin for each byte of the data bus. Even parity is generated or checked by the parity generators/checkers. Even parity means that there are an even number of HIGH inputs on the eight corresponding data bus pins and parity pin.

Data parity is generated on all write data cycles with the same timing as the data driven by the Intel486 SX microprocessor/Intel OverDrive Processor. Even parity information must be driven back to the Intel486 SX microprocessor/Intel OverDrive Processor on these pins with the same timing as read information to insure that the correct parity check status is indicated by the Intel486 SX microprocessor/Intel OverDrive Processor.

The values read on these pins do not affect program execution. It is the responsibility of the system to take appropriate actions if a parity error occurs.

Input signals on DP0–DP3 must meet setup and hold times t_{22} and t_{23} for proper operation.

Parity Status Output (PCHK#)

Parity status is driven on the PCHK# pin, and a parity error is indicated by this pin being LOW. PCHK# is driven the clock after ready for read operations to indicate the parity status for the data sampled at the end of the previous clock. Parity is checked during code reads, memory reads and I/O reads. Parity is not checked during interrupt acknowledge cycles. PCHK# only checks the parity status for enabled bytes as indicated by the byte enable and bus size signals. It is valid only in the clock immediately after read data is returned to the Intel486 SX microprocessor/Intel OverDrive Processor. At all other times it is inactive (HIGH). PCHK# is never floated.

Driving PCHK# is the only effect that bad input parity has on the Intel486 SX microprocessor/Intel OverDrive Processor. The Intel486 SX microprocessor/Intel OverDrive Processor will not vector to a bus error interrupt when bad data parity is returned. In systems that will not employ parity, PCHK# can be ignored. In systems not using parity, DP0–DP3 should be connected to V_{CC} through a pullup resistor.

6.2.5 BUS CYCLE DEFINITION

M/IO#, D/C#, W/R# Outputs

M/IO#, D/C# and W/R# are the primary bus cycle definition signals. They are driven valid as the ADS# signal is asserted. M/IO# distinguishes between memory and I/O cycles, D/C# distinguishes between data and control cycles and W/R# distinguishes between write and read cycles.

Bus cycle definitions as a function of M/IO#, D/C# and W/R# are given in Table 6.1. Note there is a difference between the Intel486 SX microprocessor/Intel OverDrive Processor and Intel386 microprocessor bus cycle definitions. The halt bus cycle type has been moved to location 001 in the Intel486 SX microprocessor/Intel OverDrive Processor from location 101 in the Intel386 microprocessor. Location 101 is now reserved and will never be generated by the Intel486 SX microprocessor/Intel OverDrive Processor.

Table 6.1. AD5# Initiated Bus Cycle Definitions

M/IO#	D/C#	W/R#	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Halt/Special Cycle
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Code Read
1	0	1	Reserved
1	1	0	Memory Read
1	1	1	Memory Write

Special bus cycles are discussed in Section 7.2.11.

Bus Lock Output (LOCK#)

LOCK# indicates that the Intel486 SX microprocessor/Intel OverDrive Processor is running a read-modify-write cycle where the external bus must not be relinquished between the read and write cycles. Read-modify-write cycles are used to implement memory-based semaphores. Multiple reads or writes can be locked.

When LOCK# is asserted, the current bus cycle is locked and the Intel486 SX microprocessor/Intel OverDrive Processor should be allowed exclusive access to the system bus. LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after ready is returned indicating the last locked bus cycle.

The Intel486 SX microprocessor/Intel OverDrive Processor will not acknowledge bus hold when LOCK# is asserted (though it will allow an address hold). LOCK# is active LOW and is floated during bus hold. Locked read cycles will not be transformed

into cache fill cycles if KEN# is returned active. Refer to Section 7.2.6 for a detailed discussion of Locked bus cycles.

Pseudo-Lock Output (PLOCK#)

The pseudo-lock feature allows atomic reads and writes of memory operands greater than 32 bits. These operands require more than one cycle to transfer. The Intel486 SX microprocessor asserts PLOCK# during segment table descriptor reads (64 bits) and cache line fills (128 bits).

> The Intel OverDrive Processor asserts PLOCK# during floating point long reads and writes (64 bits), segment table descriptor reads (64 bits) and cache line fills (128 bits).

When PLOCK# is asserted no other master will be given control of the bus between cycles. A bus hold request (HOLD) is not acknowledged during pseudo-locked reads and writes, with one exception. During non-cacheable non-bursted code prefetches, HOLD is recognized on memory cycle boundaries even though PLOCK# is asserted. The Intel486 SX microprocessor/Intel OverDrive Processor will drive PLOCK# active until the addresses for the last bus cycle of the transaction have been driven regardless of whether BRDY# or RDY# are returned.

A pseudo-locked transfer is meaningful only if the memory operand is aligned and if its completely contained within a single cache line.

> A 64-bit floating point number must be aligned to an 8-byte boundary to guarantee an atomic access.
>
> Normally PLOCK# and BLAST# are inverse of each other. However during the first cycle of a 64-bit floating point write, both PLOCK# and BLAST# will be asserted.

Since PLOCK# is a function of the bus size and KEN# inputs, PLOCK# should be sampled only in the clock ready is returned. This pin is active LOW and is not driven during bus hold. Refer to Section 7.2.7 for a detailed discussion of pseudo-locked bus cycles.

6.2.6 BUS CONTROL

The bus control signals allow the Intel486 SX microprocessor/Intel OverDrive Processor to indicate when a bus cycle has begun, and allow other system hardware to control burst cycles, data bus width and bus cycle termination.

Address Status Output (ADS#)

The ADS# output indicates that the address and bus cycle definition signals are valid. This signal will go active in the first clock of a bus cycle and go inactive in the second and subsequent clocks of the cycle. ADS# is also inactive when the bus is idle.

ADS# is used by external bus circuitry as the indication that the Intel486 SX microprocessor/Intel OverDrive Processor has started a bus cycle. The external circuit must sample the bus cycle definition pins on the next rising edge of the clock after ADS# is driven active.

ADS# is active LOW and is not driven during bus hold.

Non-burst Ready Input (RDY#)

RDY# indicates that the current bus cycle is complete. In response to a read, RDY# indicates that the external system has presented valid data on the data pins. In response to a write request, RDY# indicates that the external system has accepted the Intel486 SX microprocessor/Intel OverDrive Processor data. RDY# is ignored when the bus is idle and at the end of the first clock of the bus cycle. Since RDY# is sampled during address hold, data can be returned to the processor when AHOLD is active.

RDY# is active LOW, and is not provided with an internal pullup resistor. This input must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.

6.2.7 BURST CONTROL

Burst Ready Input (BRDY#)

BRDY# performs the same function during a burst cycle that RDY# performs during a non-burst cycle. BRDY# indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted the Intel486 SX microprocessor/Intel OverDrive Processor data in response to a write. BRDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle.

During a burst cycle, BRDY# will be sampled each clock, and if active, the data presented on the data bus pins will be strobed into the Intel486 SX microprocessor/Intel OverDrive Processor. ADS# is negated during the second through last data cycles in the burst, but address lines A2–A3 and byte enables will change to reflect the next data item expected by the Intel486 SX microprocessor/Intel OverDrive Processor.

If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is prematurely aborted. An additional complete bus cycle will be

initiated after an aborted burst cycle if the cache line fill was not complete. BRDY# is treated as a normal ready for the last data cycle in a burst transfer or for non-burstable cycles. Refer to Section 7.2.2 for burst cycle timing.

BRDY# is active LOW and is provided with a small internal pullup resistor. BRDY# must satisfy the set-up and hold times t_{16} and t_{17}.

Burst Last Output (BLAST#)

BLAST# indicates that the next time BRDY# is returned it will be treated as a normal RDY#, terminating the line fill or other multiple-data-cycle transfer. BLAST# is active for all bus cycles regardless of whether they are cacheable or not. This pin is active LOW and is not driven during bus hold.

6.2.8 INTERRUPT SIGNALS (RESET, INTR, NMI)

The interrupt signals can interrupt or suspend execution of the processor's current instruction stream.

Reset Input (RESET)

RESET forces the Intel486 SX microprocessor/Intel OverDrive Processor to begin execution at a known state. For a power-up (cold start) reset, V_{CC} and CLK must reach their proper DC and AC specifications for at least 1 ms before the Intel486 SX microprocessor/Intel OverDrive Processor begins instruction execution. The RESET pin should remain active during this time to ensure proper Intel486 SX microprocessor/Intel OverDrive Processor operation. However, for a warm boot-up case, RESET is required to remain active for a minimum of 15 clocks. The testability operating modes are programmed by the falling (inactive going) edge of RESET. (Refer to Section 8.0 for a description of the test modes during reset.)

Maskable Interrupt Request Input (INTR)

INTR indicates that an external interrupt has been generated. Interrupt processing is initiated if the IF flag is active in the EFLAGS register.

The Intel486 SX microprocessor/Intel OverDrive Processor will generate two locked interrupt acknowledge bus cycles in response to asserting the INTR pin. An 8-bit interrupt number will be latched from an external interrupt controller at the end of the second interrupt acknowledge cycle. INTR must remain active until the interrupt acknowledges have been performed to assure program interruption. Refer to Section 7.2.10 for a detailed discussion of interrupt acknowledge cycles.

The INTR pin is active HIGH and is not provided with an internal pulldown resistor. INTR is asynchronous, but the INTR setup and hold times, t_{20} and t_{21}, must be met to assure recognition on any specific clock.

Non-maskable Interrupt Request Input (NMI)

NMI is the non-maskable interrupt request signal. Asserting NMI causes an interrupt with an internally supplied vector value of 2. External interrupt acknowledge cycles are not generated since the NMI interrupt vector is internally generated. When NMI processing begins, the NMI signal will be masked internally until the IRET instruction is executed.

NMI is rising edge sensitive after internal synchronization. NMI must be held LOW for at least four CLK periods before this rising edge for proper operation. NMI is not provided with an internal pulldown resistor. NMI is asynchronous but setup and hold times, t_{20} and t_{21} must be met to assure recognition on any specific clock.

6.2.9 BUS ARBITRATION SIGNALS

This section describes the mechanism by which the processor relinquishes control of its local bus when requested by another bus master.

Bus Request Output (BREQ)

The Intel486 SX microprocessor/Intel OverDrive Processor asserts BREQ whenever a bus cycle is pending internally. Thus, BREQ is always asserted in the first clock of a bus cycle, along with ADS#. Furthermore, if the Intel486 SX microprocessor/Intel OverDrive Processor is currently not driving the bus (due to HOLD, AHOLD, or BOFF#), BREQ is asserted in the same clock that ADS# would have been asserted if the Intel486 SX microprocessor/Intel OverDrive Processor were driving the bus. After the first clock of the bus cycle, BREQ may change state. It will be asserted if additional cycles are necessary to complete a transfer (via BS8#, BS16#, KEN#), or if more cycles are pending internally. However, if no additional cycles are necessary to complete the current transfer, BREQ can be negated before ready comes back for the current cycle. External logic can use the BREQ signal to arbitrate among multiple processors. This pin is driven regardless of the state of bus hold or address hold. BREQ is active HIGH and is never floated. During a hold state, internal events may cause BREQ to be deasserted prior to any bus cycles.

Bus Hold Request Input (HOLD)

HOLD allows another bus master complete control of the Intel486 SX microprocessor/Intel OverDrive Processor bus. The Intel486 SX microprocessor/Intel OverDrive Processor will respond to an active HOLD signal by asserting HLDA and placing most of its output and input/output pins in a high impedance state (floated) after completing its current bus cycle, burst cycle, or sequence of locked cycles. In addition, if the Intel486 SX microprocessor/Intel OverDrive Processor receives a HOLD request while performing a code fetch, and that cycle is backed

off (BOFF#), the Intel486 SX microprocessor/Intel OverDrive Processor will recognize HOLD before restarting the cycle. The BREQ, HLDA, PCHK# and FERR# pins are not floated during bus hold. The Intel486 SX microprocessor/Intel OverDrive Processor will maintain its bus in this state until the HOLD is deasserted. Refer to Section 7.2.9 for timing diagrams for bus hold cycles and HOLD request acknowledge during BOFF#.

Unlike the Intel386 microprocessor, the Intel486 SX microprocessor/Intel OverDrive Processor will recognize HOLD during reset. Pullup resistors are not provided for the outputs that are floated in response to HOLD. HOLD is active HIGH and is not provided with an internal pulldown resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper chip operation.

Bus Hold Acknowledge Output (HLDA)

HLDA indicates that the Intel486 SX microprocessor/Intel OverDrive Processor has given the bus to another local bus master. HLDA goes active in response to a hold request presented on the HOLD pin. HLDA is driven active in the same clock that the Intel486 SX microprocessor/Intel OverDrive Processor floats its bus.

HLDA will be driven inactive when leaving bus hold and the Intel486 SX microprocessor/Intel OverDrive Processor will resume driving the bus. The Intel486 SX microprocessor/Intel OverDrive Processor will not cease internal activity during bus hold since the internal cache will satisfy the majority of bus requests. HLDA is active HIGH and remains driven during bus hold.

Backoff Input (BOFF#)

Asserting the BOFF# input forces the Intel486 SX microprocessor/Intel OverDrive Processor to release control of its bus in the next clock. The pins floated are exactly the same as in response to HOLD. The response to BOFF# differs from the response to HOLD in two ways: First, the bus is floated immediately in response to BOFF# while the Intel486 SX microprocessor/Intel OverDrive Processor completes the current bus cycle before floating its bus in response to HOLD. Second the Intel486 SX microprocessor/Intel OverDrive Processor does not assert HLDA in response to BOFF#.

The Intel486 SX microprocessor/Intel OverDrive Processor remains in bus hold until BOFF# is negated. Upon negation, the Intel486 SX microprocessor/Intel OverDrive Processor restarts the bus cycle aborted when BOFF# was asserted. To the internal execution engine the effect of BOFF# is the same as inserting a few wait states to the original cycle. Refer to Section 7.2.12 for a description of bus cycle restart.

Any data returned to the Intel486 SX microprocessor/Intel OverDrive Processor while BOFF# is asserted is ignored. BOFF# has higher priority than RDY# or BRDY#. If both BOFF# and ready are returned in the same clock, BOFF# takes effect. If BOFF# is asserted while the bus is idle, the Intel486 SX microprocessor/Intel OverDrive Processor will float its bus in the next clock. BOFF# is active LOW and must meet setup and hold times t_{18} and t_{19} for proper chip operation.

6.2.10 CACHE INVALIDATION

The AHOLD and EADS# inputs are used during cache invalidation cycles. AHOLD conditions the Intel486 SX microprocessor/Intel OverDrive Processor address lines, A4–A31, to accept an address input. EADS# indicates that an external address is actually valid on the address inputs. Activating EADS# will cause the Intel486 SX microprocessor/Intel OverDrive Processor to read the external address bus and perform an internal cache invalidation cycle to the address indicated. Refer to Section 7.2.8 for cache invalidation cycle timing.

Address Hold Request Input (AHOLD)

AHOLD is the address hold request. It allows another bus master access to the Intel486 SX microprocessor/Intel OverDrive Processor address bus for performing an internal cache invalidation cycle. Asserting AHOLD will force the Intel486 SX microprocessor/Intel OverDrive Processor to stop driving its address bus in the next clock. While AHOLD is active only the address bus will be floated, the remainder of the bus can remain active. For example, data can be returned for a previously specified bus cycle when AHOLD is active. The Intel486 SX microprocessor/Intel OverDrive Processor will not initiate another bus cycle during address hold. Since the Intel486 SX microprocessor/Intel OverDrive Processor floats its bus immediately in response to AHOLD, an address hold acknowledge is not required. If AHOLD is asserted while a bus cycle is in progress, and no readies are returned during the time AHOLD is asserted, the Intel486 SX microprocessor/Intel OverDrive Processor will redrive the same address (that it originally sent out) once AHOLD is negated.

AHOLD is recognized during reset. Since the entire cache is invalidated by reset, any invalidation cycles run during reset will be unnecessary. AHOLD is active HIGH and is provided with a small internal pulldown resistor. It must satisfy the setup and hold times t_{18} and t_{19} for proper chip operation. This pin determines whether or not the built in self test features of the Intel486 SX microprocessor/Intel OverDrive Processor will be exercised on assertion of RESET.

External Address Valid Input (EADS#)

EADS# indicates that a valid external address has been driven onto the Intel486 SX microprocessor/ Intel OverDrive Processor address pins. This address will be used to perform an internal cache invalidation cycle. The external address will be checked with the current cache contents. If the address specified matches any areas in the cache, that area will immediately be invalidated.

An invalidation cycle may be run by asserting EADS# regardless of the state of AHOLD, HOLD and BOFF#. EADS# is active LOW and is provided with an internal pullup resistor. EADS# must satisfy the setup and hold times t_{12} and t_{13} for proper chip operation.

6.2.11 CACHE CONTROL

Cache Enable Input (KEN#)

KEN# is the cache enable pin. KEN# is used to determine whether the data being returned by the current cycle is cacheable. When KEN# is active and the Intel486 SX microprocessor/Intel OverDrive Processor generates a cycle that can be cached (most any memory read cycle), the cycle will be transformed into a cache line fill cycle.

A cache line is 16 bytes long. During the first cycle of a cache line fill the byte-enable pins should be ignored and data should be returned as if all four byte enables were asserted. The Intel486 SX microprocessor/Intel OverDrive Processor will run between 4 and 16 contiguous bus cycles to fill the line depending on the bus data width selected by BS8# and BS16#. Refer to Section 7.2.3 for a description of cache line fill cycles.

The KEN# input is active LOW and is provided with a small internal pullup resistor. It must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

Cache Flush Input (FLUSH#)

The FLUSH# input forces the Intel486 SX microprocessor/Intel OverDrive Processor to flush its entire internal cache. FLUSH# is active LOW and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition on any specific clock.

FLUSH# also determines whether or not the tristate test mode of the Intel486 SX microprocessor/Intel OverDrive Processor will be invoked on assertion of RESET. To initiate the tri-state test mode, FLUSH# should be active for 2 clks before and 2 clks after the reset is deasserted. However, if FLUSH# is asserted synchronously keeping it active for 2 CLKs prior to the falling edge of RESET will initiate the tri-state mode.

6.2.12 PAGE CACHEABILITY (PWT, PCD)

The PWT and PCD output signals correspond to two user attribute bits in the page table entry. When paging is enabled, PWT and PCD correspond to bits 3 and 4 of the page table entry respectively. For cycles that are not paged when paging is enabled (for example I/O cycles) PWT and PCD correspond to bits 3 and 4 in control register 3. When paging is disabled, the Intel486 SX microprocessor/Intel OverDrive Processor ignores the PCD and PWT bits and assumes they are zero for the purpose of caching and driving PCD and PWT.

PCD is masked by the CD (cache disable) bit in control register 0 (CR0). When CD = 1 (cache line fills disabled) the Intel486 SX microprocessor/Intel OverDrive Processor forces PCD HIGH. When CD = 0, PCD is driven with the value of the page table entry/directory.

The purpose of PCD is to provide a cacheable/non-cacheable indication on a page by page basis. The Intel486 SX microprocessor/Intel OverDrive Processor will not perform a cache fill to any page in which bit 4 of the page table entry is set. PWT corresponds to the write-back bit and can be used by an external cache to provide this functionality. PCD and PWT bits are assigned to be zero during real mode or whenever paging is disabled. Refer to Sections 4.5.4 and 5.6 for a discussion of non-cacheable pages.

PCD and PWT have the same timing as the cycle definition pins (M/IO#, D/C#, W/R#). PCD and PWT are active HIGH and are not driven during bus hold.

6.2.13.1 Intel OverDrive Processor/Intel487 SX Math CoProcessor Signals

Upgrade Present (UP#)

The UP# pin is present on the Intel OverDrive Processor/Intel487 SX Math CoProcessor only. The Intel OverDrive Processor/Intel487 SX Math CoProcessor uses UP# to signal to the Intel486 SX microprocessor that an Intel OverDrive Processor/Intel487 SX Math CoProcessor is present in the system. An UP# driven interface circuit floats the Intel486 SX microprocessor/Intel OverDrive Processor/Intel487 SX Math CoProcessor then takes charge of the bus. The Intel OverDrive Processor/Intel487 SX Math CoProcessor always drive UP# active. UP# will never be floated if an Intel OverDrive Processor/Intel487 SX Math CoProcessor is present in the system. Since the Intel OverDrive Processor/Intel487 SX Math CoProcessor always drives UP# active, UP# can be used to check whether the system contains an Intel OverDrive Processor/ Intel487 SX Math CoProcessor.

Key (Key)

The Key pin is electrically nonfunctional. Its only purpose is to insure correct Intel OverDrive Processor/Intel487 SX Math CoProcessor orientation in a 169-pin socket.

6.2.13.2 Numeric Error Reporting (FERR#, IGNNE#)

To allow PC-type floating point error reporting, the Intel OverDrive Processor provides two pins, FERR# and IGNNE#.

Floating Point Error Output (FERR#)

The Intel OverDrive Processor asserts FERR# whenever an unmasked floating point error is encountered. FERR# is similar to the ERROR# pin on the Intel387 Math CoProcessor. FERR# can be used by external logic for PC-type floating point error reporting in systems with an Intel OverDrive Processor. FERR# is active LOW, and is not floated during bus hold.

In some cases, FERR# is asserted when the next floating point instruction is encountered and in other cases it is asserted before the next floating point instruction is encountered depending upon the execution state of the instruction causing the exception.

The following class of floating point exceptions drive FERR# at the time the exception occurs (i.e., before encountering the next floating point instruction).

1. The stack fault, invalid operation, and denormal exceptions on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exceptions on store instructions (including integer store instructions).

The following class of floating point exceptions drive FERR# only after encountering the next floating point instruction.

1. Exceptions other than on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exception on all basic arithmetic, load, compare, and control instructions (i.e., all other instructions).

Ignore Numeric Error Input (IGNNE#)

The Intel OverDrive Processor will ignore a numeric error and continue executing non-control floating point instructions when IGNNE# is asserted, but FERR# will still be activated. When deasserted, the Intel OverDrive Processor will freeze on a non-control floating point instruction if a previous instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set.

The IGNNE# input is active LOW and is provided with a small internal pullup resistor. This input is asynchronous, but must meet setup and hold times t_{20} and t_{21} to insure recognition on any specific clock.

6.2.14 BUS SIZE CONTROL (BS16#, BS8#)

The BS16# and BS8# inputs allow external 16- and 8-bit busses to be supported with a small number of external components. The Intel486 SX microprocessor/Intel OverDrive Processor samples these pins every clock. The value sampled in the clock before ready determines the bus size. When asserting BS16# or BS8# only 16 or 8 bits of the data bus need be valid. If both BS16# and BS8# are asserted, an 8-bit bus width is selected.

When BS16# or BS8# are asserted the Intel486 SX microprocessor/Intel OverDrive Processor will convert a larger data request to the appropriate number of smaller transfers. The byte enables will also be modified appropriately for the bus size selected.

BS16# and BS8# are active LOW and are provided with small internal pullup resistors. BS16# and BS8# must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

6.2.15 ADDRESS BIT 20 MASK (A20M#)

Asserting the A20M# input causes the Intel486 SX microprocessor/Intel OverDrive Processor to mask physical address bit 20 before performing a lookup in the internal cache and before driving a memory cycle to the outside world. When A20M# is asserted, the Intel486 SX microprocessor/Intel OverDrive Processor emulates the 1 Mbyte address wrap-around that occurs on the 8086. A20M# is active LOW and must be asserted only when the processor is in real mode. The A20M# is not defined in Protected Mode. A20M# is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For correct operation of the chip, A20M# should not be active at the falling edge of RESET. When A20M# is asserted asynchronously, A20M# should be high (non-active) for 2 clocks before and 2 clocks after RESET is deasserted. When A20M# is asserted synchronously, A20M# should be high (non-active) at the clock prior to the falling edge of RESET.

A20M# exhibits a minimum 4 clock latency, from time of assertion to masking of the A20 bit. A20M# is ignored during cache invalidation cycles. I/O

writes require A20M# to be asserted a minimum of 2 clocks prior to RDY being returned for the I/O write. This insures recognition of the address mask before the i486 SX microprocesor/Intel OverDrive Processor begins execution of the instruction following OUT. If A20M# is asserted after the ADS# of a data cycle, the A20 address signal is not masked during this cycle but is masked in the next cycle. During a prefetch (cacheable or not), if A20M# is asserted after the first ADS#, A20 is not masked for the duration of the prefetch; even if BS16# ro BS8# is asserted.

6.2.16 PERFORMANCE UPGRADE SUPPORT SIGNAL DESCRIPTION (Available on PQFP Version Only)

Upgrade Present Input (UP#)

UP# is an input for the Intel486 SX microprocessor and is sampled at all times. UP# indicates that the performance upgrade component is present in the system. It is driven by the MP# pin of the Intel OverDrive Processor. UP# forces the Intel486 SX CPU to tri-state all its outputs and enter the power-down mode. In the power down mode the Intel486 SX CPU keeps its outputs tri-stated and reduces its power dissipation to a minimal value (see DC specifications).

6.2.17 BOUNDARY SCAN TEST SIGNALS

The following boundary scan test signals are only available on the PQFP version of the Intel486 SX CPU.

Test Clock (TCK)

TCK is an input to the Intel486™ SX CPU and provides the clocking function required by the JTAG boundary scan feature. TCK is used to clock state information and data into and out of the component. State select information and data are clocked into the component on the rising edge of TCK on TMS and TDI, respectively. Data is clocked out of the part on the falling edge of TCK on TDO.

In addition to using TCK as a free running clock, it may be stopped in a low, O, state, indefinitely as described in IEEE 1149.1. While TCK is stopped in the low state, the boundary scan latches retain their state.

When boundary scan is not used, TCK should be tied high or left as a NC. (This is important during power up to avoid the possibility of glitches on the TCK which could prematurely initiate boundary scan operations.) TCK is supplied with an internal pullup resistor.

TCK is a clock signal and is used as a reference for sampling other JTAG signals. On the rising edge of TCK, TMS and TDI are sampled. On the falling edge of TCK, TDO is driven.

Test Mode Select (TMS)

TMS is decoded by the JTAG TAP (Tap Access Port) to select the operation of the test logic, as described in Section 5.2.4.

To guarantee deterministic behavior of the TAP controller TMS is provided with an internal pull-up resistor. If boundary scan is not used, TMS may be tied high or left unconnected. TMS is sampled on the rising edge of TCK. TMS is used to select the internal TAP states required to load boundary scan instructions to data on TDI. For proper initialization of the JTAG logic, TMS should be driven high, "1", for at least four TCK cycles following the rising edge of RESET.

Test Data Input (TDI)

TDI is the serial input used to shift JTAG instructions and data into the component. The shifting of instructions and data occurs during the SHIFT-IR and SHIFT-DR TAP controller states, respectively. These states are selected using the TMS signal as described in Section 5.2.4.

An internal pullup resistor is provided on TDI to ensure a known logic state if an open circuit occurs on the TDI path. Note that when "1" is continuously shifted into the instruction register, the BYPASS instruction is selected. TDI is sampled on the rising edge of TCK, during the SHIFT-IR and the SHIFT-DR states. During all other TAP controller states, TDI is a "don't care". TDI is only sampled when TMS and TCK have been used to select the SHIFT-IR or SHIFT-DR states in the TAP controller. For proper initialization of JTAG logic, TDI should be driven high, "1", for at least four TCK cycles following the rising edge of RESET.

Test Data Output (TDO)

TDO is the serial output used to shift JTAG instructions and data out of the component. The shifting of instructions and data occurs during the SHIFT-IR and SHIFT-DR TAP controller states, respectively. These states are selected using the TMS signal as described in Section 5.2.4. When not in SHIFT-IR or SHIFT-DR state, TDO is driven to a high impedance state to allow connecting TDO of different devices in parallel. TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT-DR TAP controller states. At all other times TDO is driven to the high impedance state. TDO is only driven when TMS and

TCK have been used to select the SHIFT-IR or SHIFT-DR states in the TAP controller.

6.3 Write Buffers

The Intel486 SX microprocessor/Intel OverDrive Processor contain four write buffers to enhance the performance of consecutive writes to memory. The buffers can be filled at a rate of one write per clock until all four buffers are filled.

When all four buffers are empty and the bus is idle, a write request will propagate directly to the external bus bypassing the write buffers. If the bus is not available at the time the write is generated internally, the write will be placed in the write buffers and propagate to the bus as soon as the bus becomes available. The write is stored in the on-chip cache immediately if the write is a cache hit.

Writes will be driven onto the external bus in the same order in which they are received by the write buffers. Under certain conditions a memory read will go onto the external bus before the memory writes pending in the buffer even though the writes occurred earlier in the program execution.

A memory read will only be reordered in front of all writes in the buffers under the following conditions: If all writes pending in the buffers are cache hits and the read is a cache miss. Under these conditions the Intel486 SX microprocessor/Intel OverDrive Processor will not read from an external memory location that needs to be updated by one of the pending writes.

Reordering of a read with the writes pending in the buffers can only occur once before all the buffers are emptied. Reordering read once only maintains cache consistency. Consider the following example: The Intel486 SX microprocessor/Intel OverDrive Processor writes to location X. Location X is in the internal cache, so it is updated there immediately. However, the bus is busy so the write out to main memory is buffered (see Figure 6.3(a)). At this point, any reads to location X would be cache hits and most up-to-date data would be read.

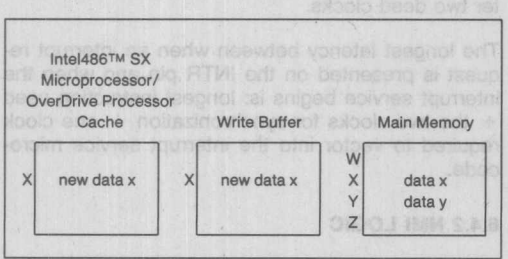


Figure 6.3(a). Internal Cache Example

The next instruction causes a read to location Y. Location Y is not in the cache (a cache miss). Since the write in the write buffer is a cache hit, the read is reordered. When location Y is read, it is put into the cache. The possibility exists that location Y will replace location X in the cache. If this is true, location X would no longer be cached (see Figure 6.3(b)).

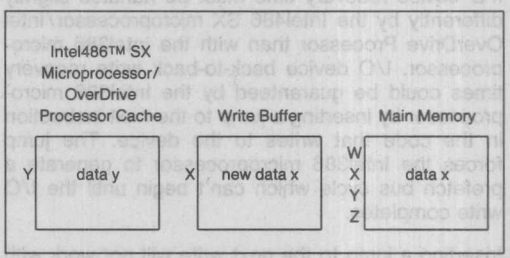


Figure 6.3(b). Internal Cache Example

Cache consistency has been maintained up to this point. If a subsequent read is to location X (now a cache miss) and it was reordered in front of the buffered write to location X, stale data would be read. This is why only 1 read is allowed to be reordered. Once a read is reordered, all the writes in the write buffer are flagged as cache misses to ensure that no more reads are reordered. Since one of the conditions to reorder a read is that all writes in the write buffer must be cache hits, no more reordering is allowed until all of those flagged writes propogate to the bus. Similarly, if an invalidation cycle is run all entries in the write buffer are flagged as cache misses.

For multiple processor systems and/or systems using DMA techniques, such as bus snooping, locked semaphores should be used to maintain cache consistency.

6.3.1 WRITE BUFFERS AND I/O CYCLES

Input/Output (I/O) cycles must be handled in a different manner by the write buffers.

I/O reads are never reordered in front of buffered memory writes. This insures that the Intel486 SX microprocessor/Intel OverDrive Processor will update all memory locations before reading status from an I/O device.

The Intel486 SX microprocessor/Intel OverDrive Processor never buffers single I/O writes. When processing an OUT instruction, internal execution stops until the I/O write actually completes on the external bus. This allows time for the external sys-

tem to drive an invalidate into the Intel486 SX microprocessor/Intel OverDrive Processor or to mask interrupts before the Intel486 SX microprocessor/Intel OverDrive Processor progresses to the instruction following OUT. REP OUTS instructions will be buffered.

I/O device recovery time must be handled slightly differently by the Intel486 SX microprocessor/Intel OverDrive Processor than with the Intel386 microprocessor. I/O device back-to-back write recovery times could be guaranteed by the Intel386 microprocessor by inserting a jump to the next instruction in the code that writes to the device. The jump forces the Intel386 microprocessor to generate a prefetch bus cycle which can't begin until the I/O write completes.

Inserting a jump to the next write will not work with the Intel486 SX microprocessor/Intel OverDrive Processor because the prefetch could be satisfied by the on-chip cache. A read cycle must be explicitly generated to a non-cacheable location in memory to guarantee that a read bus cycle is performed. This read will not be allowed to proceed to the bus until after the I/O write has completed because I/O writes are not buffered. The I/O device will have time to recover to accept another write during the read cycle.

6.3.2 WRITE BUFFERS IMPLICATIONS ON LOCKED BUS CYCLES

Locked bus cycles are used for read-modify-write accesses to memory. During a read-modify-write access, a memory base variable is read, modified and then written back to the same memory location. It is important that no other bus cycles, generated by other bus masters or by the Intel486 SX microprocessor/Intel OverDrive Processor be allowed on the external bus between the read and write portion of the locked sequence.

During a locked read cycle the Intel486 SX microprocessor/Intel OverDrive Processor will always access external memory, it will never look for the location in the on-chip cache, but for write cycles, data is written in the internal cache (if cache hit) and in the external memory. All data pending in the Intel486 SX microprocessor/Intel OverDrive Processor write buffers will be written to memory before a locked cycle is allowed to proceed to the external bus.

The Intel486 SX microprocessor/Intel OverDrive Processor will assert the LOCK# pin after the write buffers are emptied during a locked bus cycle. With the LOCK# pin asserted, the Intel486 SX microprocessor/Intel OverDrive Processor will read the data, operate on the data and place the results in a write buffer. The contents of the write buffer will then be written to external memory. LOCK# will become inactive after the write part of the locked cycle.

6.4 Interrupt and Non-Maskable Interrupt Interface

The Intel486 SX microprocessor/Intel OverDrive Processor provide two asynchronous interrupt inputs, INTR (interrupt request) and NMI (non-maskable interrupt input). This section describes the hardware interface between the instruction execution unit and the pins. For a description of the algorithmic response to interrupts refer to Section 2.7. For interrupt timings refer to Section 7.2.10.

6.4.1 INTERRUPT LOGIC

The Intel486 SX microprocessor/Intel OverDrive Processor contain a two-clock synchronizer on the interrupt line. An interrupt request will reach the internal instruction execution unit two clocks after the INTR pin is asserted, if proper setup is provided to the first stage of the synchronizer. There is no special logic in the interrupt path other than the synchronizer. The INTR signal is level sensitive and must remain active for the instruction execution unit to recognize it. The interrupt will not be serviced by the Intel486 SX microprocessor/Intel OverDrive Processor if the INTR signal does not remain active.

The instruction execution unit will look at the state of the synchronized interrupt signal at specific clocks during the execution of instructions (if interrupts are enabled). These specific clocks are at instruction boundaries, or iteration boundaries in the case of string move instructions. Interrupts will only be accepted at these boundaries.

An interrupt must be presented to the Intel486 SX microprocessor/Intel OverDrive Processor INTR pin three clocks before the end of an instruction for the interrupt to be acknowledged. Presenting the interrupt 3 clocks before the end of an instruction allows the interrupt to pass through the two clock synchronizer leaving one clock to prevent the initiation of the next sequential instruction and to begin interrupt service. If the interrupt is not received in time to prevent the next instruction, it will be accepted at the end of next instruction, assuming INTR is still held active. The interrupt service microcode will start after two dead clocks.

The longest latency between when an interrupt request is presented on the INTR pin and when the interrupt service begins is: longest instruction used + the two clocks for synchronization + one clock required to vector into the interrupt service microcode.

6.4.2 NMI LOGIC

The NMI pin has a synchronizer like that used on the INTR line. Other than the synchronizer, the NMI logic is different from that of the maskable interrupt.

NMI is edge triggered as opposed to the level triggered INTR signal. The rising edge of the NMI signal is used to generate the interrupt request. The NMI input need not remain active until the interrupt is actually serviced. The NMI pin only needs to remain active for a single clock if the required setup and hold times are met. NMI will operate properly if it is held active for an arbitrary number of clocks.

The NMI input must be held inactive for at least four clocks after it is asserted to reset the edge triggered logic. A subsequent NMI may not be generated if the NMI is not held inactive for at least two clocks after being asserted.

The NMI input is internally masked whenever the NMI routine is entered. The NMI input will remain masked until an IRET (return from interrupt) instruction is executed. Masking the NMI signal prevents recursive NMI calls. If another NMI occurs while the NMI is masked off, the pending NMI will be executed after the current NMI is done. Only one NMI can be pending while NMI is masked.

6.5 Reset and Initialization

The Intel486 SX microprocessor has a built in self test (BIST) that can be run during reset. The BIST is invoked if the AHOLD pin is asserted for 2 clocks before and 2 clocks after RESET is deasserted. RESET must be active for 15 clocks with or with no BIST being enabled. To ensure proper results, FLUSH# must not be asserted while BIST is executing. Refer to Section 8.0 for information on Intel486 SX microprocessor/Intel OverDrive Processor testability.

The Intel486 SX microprocessor/Intel OverDrive Processor registers have the values shown in Table 6.2 after RESET is performed. The EAX register contains information on the success or failure of the BIST if the self test is executed. The DX register always contains a component identifier at the conclusion of RESET. The upper byte of DX (DH) will

2

Table 6.2. Register Values after Reset

Register	Initial Value (BIST)	Initial Value (No Bist)
EAX	Zero (Pass)	Undefined
ECX	Undefined	Undefined
EDX	0400 + Revision ID	0400 + Revision ID
EBX	Undefined	Undefined
ESP	Undefined	Undefined
EBP	Undefined	Undefined
ESI	Undefined	Undefined
EDI	Undefined	Undefined
EFLAGS	00000002h	00000002h
EIP	0FFF0h	0FFF0h
ES	0000h	0000h
CS	F000h*	F000h*
SS	0000h	0000h
DS	0000h	0000h
FS	0000h	0000h
GS	0000h	0000h
IDTR	Base = 0, Limit = 3FFh	Base = 0, Limit = 3FFh
CR0	60000010h	60000010h
DR7	00000000h	00000000h
CW	037Fh	Unchanged
SW	0000h	Unchanged
TW	FFFFh	Unchanged
FIP	00000000h	Unchanged
FEA	00000000h	Unchanged
FCS	0000h	Unchanged
FDS	0000h	Unchanged
FOP	000h	Unchanged
FSTACK	Undefined	Unchanged

contain 04 and the lower byte (DL) will contain a stepping identifier (see Table 6-3).

The floating point registers are initialized as if the FINIT/FNINIT (initialize processor) instruction was executed if the BIST was performed. If the BIST is not executed, the floating point registers are unchanged.

Table 6-3. Intel486™ SX Microprocessor/Intel OverDrive Processor/Intel487™ SX Math CoProcessor Revision ID

Product	Stepping	Component ID	Revision ID
i486 SX Microprocessor	A0	04	20
	B0	04	22
	cA0	04	27
	cB0	04	28
i487 SX Math CoProcessor	A0	04	20
	B0	04	21
OverDrive Processor	A2	04	32
	B1	04	33

The Intel486 SX microprocessor/Intel OverDrive Processor will start executing instructions at location FFFFFFF0H after RESET. When the first InterSegment Jump or Call is executed, address lines A20–A31 will drop LOW for CS-relative memory cycles, and the Intel486 SX microprocessor/Intel OverDrive Processor will only execute instructions in the lower one Mbyte of physical memory. This allows the system designer to use a ROM at the top of physical memory to initialize the system and take care of RESETs.

RESET forces the Intel486 SX microprocessor/Intel OverDrive Processor to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active.

All entries in the cache are invalidated by RESET.

6.5.1 PIN STATE DURING RESET

The Intel486 SX microprocessor/Intel OverDrive Processor recognizes and can respond to HOLD, AHOLD, and BOFF# requests regardless of the state of RESET. Thus, even though the processor is in reset, it can still float its bus in response to any of these requests.

While in reset, the Intel486 SX microprocessor/Intel OverDrive Processor bus is in the state shown in Figure 6.4 if the HOLD, AHOLD and BOFF# requests are inactive. The figure shows the bus state for the Intel486 SX microprocessor when the Intel OverDrive Processor is not present or for Intel OverDrive Processor when it is installed in the system. Note that the address (A31–A2, BE3#–BE0#) and cycle definition (M/IO#, D/C#, W/R#) pins are undefined from the time reset is asserted up to the start of the first bus cycle. All undefined pins (except FERR#) assume known values at the beginning of the first bus cycle. The first bus cycle is always a code fetch to address FFFFFFF0H.

FERR# reflects the state of the ES (error summary status) bit in the floating point unit status word. The ES bit is initialized whenever the floating point unit state is initialized. The floating point unit's status word register can be initialized by BIST or by executing FINIT/FNINIT instruction. Thus, after reset and before executing the first FINIT or FNINIT instruction, the values of the FERR# and the numeric status word register bits 0–7 depends on whether or not BIST is performed. Table 6-4 shows the state of FERR# signal after reset and before the execution of the FINIT/FNINIT instruction.

Table 6-4

BIST Performed	FERR# Pin	FPU Status Word Register Bits 0–7
YES	Inactive (High)	Inactive (Low)
NO	Undefined (Low or High)	Undefined (Low or High)

After the first FINIT or FNINIT instruction, FERR# pin and the FPU status word register bits (0–7) will be inactive irrrespective of the Built-In Self-Test (BIST).

The bus state of the Intel486 SX microprocessor during reset with Intel OverDrive Processor present in the system is shown in Figure 6.5. Note that BOFF# is driven active to float most of the output signals during reset and remaining are floated after the falling edge of reset on sampling FLUSH# active.

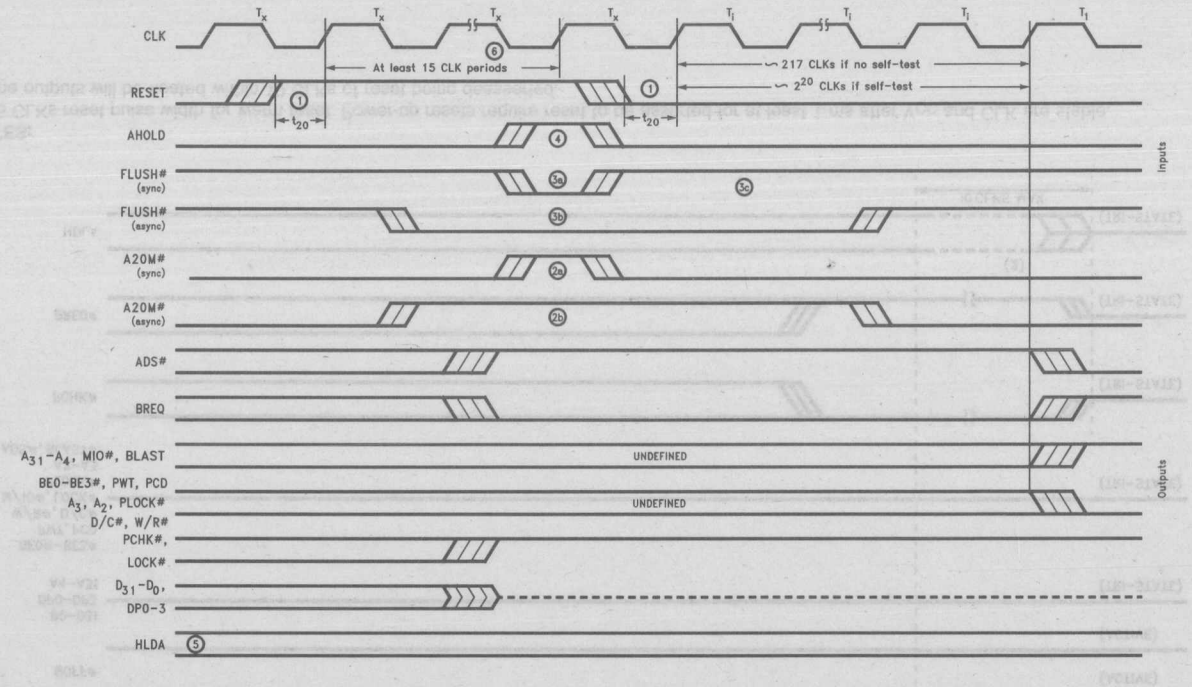


Figure 6.4. Pin States during RESET

NOTES:

1. RESET is an asynchronous input. t_{20} must be met only to guarantee recognition on a specific clock edge.
2a. When A20M# is driven synchronously, it must be driven high (inactive) for the CLK edge prior to the falling edge of RESET to ensure proper operation. A20M# setup and hold times must be met.
2b. When A20M# is driven asynchronously, it must be driven high (inactive) for two CLKs prior to and two CLKs after the falling edge of RESET to ensure proper operation.
3a. When FLUSH# is driven synchronously, it should be driven low (active) for the CLK edge prior to the falling edge of RESET to invoke the Tri-State Output Test Mode. All outputs are guaranteed tri-stated within 10 CLKs of RESET being deasserted. FLUSH# setup and hold times must be met.
3b. When FLUSH# is driven asynchronously, it must be driven low (active) for two CLKs prior to and two CLKs after the falling edge of RESET to invoke the Tri-State Output Test Mode. All outputs are guaranteed tri-stated within 10 CLKs of RESET being deasserted.
3c. FLUSH# must be driven high (inactive) during, Built-in-Self-Test (BIST).
4. AHOLD should be driven high (active) for the CLK edge prior to the falling edge of RESET to invoke the Built-In-Self-Test (BIST). AHOLD setup and hold times must be met.
5. Hold is recognized normally during RESET.
6. 15 CLKs RESET pulse width for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.

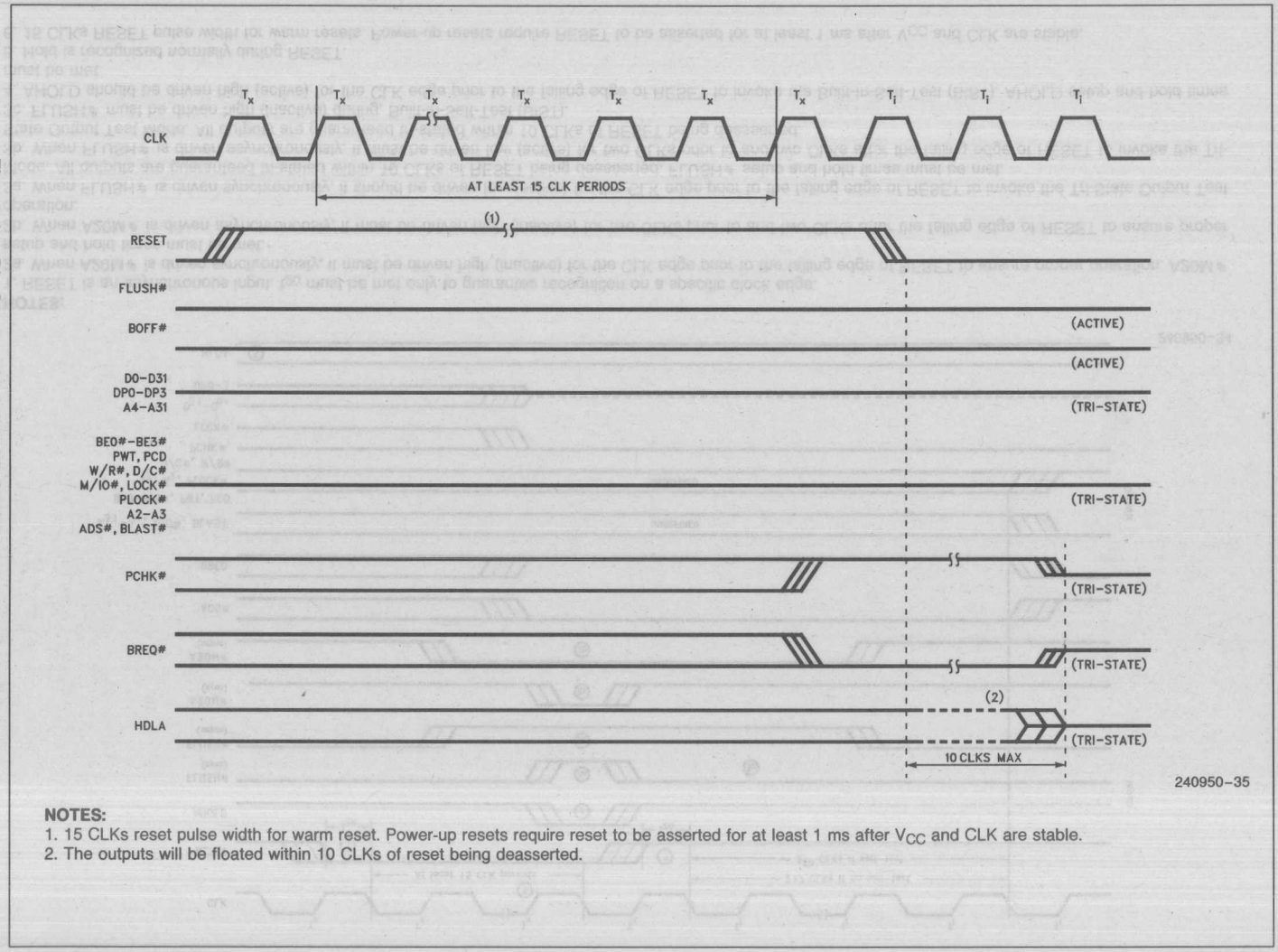


Figure 6.5. Pin States of Intel486™ SX Microprocessor in PGA, during RESET, with Intel OverDrive™ Processor in the System

NOTES:
1. 15 CLKs reset pulse width for warm reset. Power-up resets require reset to be asserted for at least 1 ms after V_{CC} and CLK are stable.
2. The outputs will be floated within 10 CLKs of reset being deasserted.

6.6 Upgrade Circuit

Figures 6.6 and 6.7 show the circuit to allow upgrades with an Intel OverDrive Processor or Intel487 SX Match CoProcessor.

6.6.1 UPGRADE CIRCUIT FOR Intel486 SX CPU IN PGA

The circuit in Figure 6.6 is recommended for Intel486 SX CPU in PGA package only. All address, data and control signals including CPU clock of the Intel486 SX microprocessor, are tied to the corresponding signals of the Intel OverDrive Processor Note that the UP# and FERR# pins in the Intel OverDrive Processor should be tied high to allow the system to function normally when the Intel OverDrive Processor is not present. Additionally, HLDA of the Intel486 SX microprocessor and the Intel OverDrive Processor should be pulled-down and ORed as shown in the Figure 6.6. This will allow HLDA to be asserted normally when this Intel OverDrive Processor is not installed in the system. This arrangement also takes care of the case of HOLD during reset in the presence of an Intel OverDrive Processor in the system. In this case the Intel486 SX microprocessor, under the influence of BOFF#, may not recognize HOLD, while the Intel OverDrive Processor will sense HOLD and assert the HLDA signal. The OR gate on the HLDA signals avoids the contention that may have occurred from this.

When the Intel OverDrive Processor is inserted in the system, the Intel OverDrive Processor UP# pin drives the BOFF# and FLUSH# pins of the Intel486 SX microprocessor active. The BOFF# signal floats most of the outputs (except PCHK#, BREQ# and HLDA) of the Intel486 SX microprocessor on the next clock, while FLUSH# is sampled at the falling edge of the reset by the Intel486 SX microprocessor and all its outputs are floated. The Intel OverDrive Processor then takes charge of the bus and the system works normally. (See Figure 6.5 for timing details during reset.)

6.6.2 UPGRADE CIRCUIT FOR Intel486 SX CPU IN PQFP

The PQFP version of the Intel486 SX CPU provides a direct interface with the OverDrive Processor. Thus the interface logic between the Intel486 SX CPU and the OverDrive Processor as shown in Figure 6.6 is not required. Instead, the UP# pin of the OverDrive Processor is connected directly to the UP# pin of the Intel486 SX CPU. All other signals are connected together as shown in Figure 6.7.

6.6.2.1 Power Down Mode

Power Down Mode is a new mode in the Intel486 SX CPU in PQFP. It is initiated by the Intel OverDrive Processor via the Upgrade Circuit. Upon sensing the presence of the Intel OverDrive Processor, Intel486 SX microprocessor not only tri-states its outputs but also enters the "Power Down Mode", whereby it remains tri-stated and lowers its power consumption (See D.C. Specfications)

The Intel486 SX CPU samples the UP# pin to enter the Power Down Mode (PDM). The UP# pin of the Intel486 SX microprocessor is driven active by the UP# pin of the Intel OverDrive Processor in the Performance Upgrade Circuit, as shown in Figure 6.7.

UP# of the OverDrive Processor is driven active after power-up. It drives the UP# pin of the Intel486 SX CPU to signal that the upgrade component has been installed in the system. Within a few clocks the Intel486 SX CPU tri-states all of its outputs. This is followed by the power down mode after the falling edge of Reset. Once the Intel486 SX CPU enters the power down mode, it will remain in it until the next reset. For warm resets, the Intel486 SX CPU will remain tri-stated and re-enter the power down mode after the reset has been de-asserted. Similarly for the power-up reset, if the Intel OverDrive Processor is not taken out of the system, the Intel486 SX microprocessor will be tri-stated and will enter Power Down Mode after the falling edge of reset.

6.6.3 DESIGN CONSIDERATIONS

The interfaces shown in Figure 6.6 and 6.7 are designed for a discrete OverDrive Processor socket. The following should be considered when designing an Intel486 SX microprocesser/Intel OverDrive Processor system.

1. The timing loops should be independent of the cpi. One way to attain this is to implement these loops in hardware and not in software (e.g., BIOS).
2. Initialization routine should check the presence of a math coprocesser e.g., Intel OverDrive Processor and should set the floating point related bits in the CR0 register accordingly. Recommended bit pattern is given in Table 6-5. The FSTCW instruction will give a value of FFFFh for the Intel486 SX microprocessor and 037Fh for the Intel OverDrive Processor.

Table 6-5. Recommended Values of the FP Related Bits for Intel486™ SX Microprocessor/Intel OverDrive Processor System

CR0 Bit	Intel486 SX Microprocessor	Intel OverDrive Processor
EM	1	0
MP	0	1
NE	1	0, for DOS Systems 1, for User-Defined Exception Handler

Following is an example code to initialize the system and check for the presence of Intel486 SX microprocessor/Intel OverDrive Processor.

```
fninit
fstcw       mem_loc
mov         ax, mem_loc
cmp         ax, 037fh
jz          Intel OverDrive Processor_present       ;ax=037fh
jmp         Intel486 SX microprocessor_present      ;ax=ffffh
```

If the Intel OverDrive Processor is not present, the following code can be run to set the CR0 register for Intel486 SX microprocessor:

```
mov         eax, cr0
and         eax, fffffffdh                          ;make MP=0
or          eax, 0024h                              ;make EM=1, NE=1
mov         cr0, eax
```

The above initialization will cause any floating point instruction to generate the interrupt 7. The software emulation will then take control to execute these instructions. This code is not required if Intel OverDrive Processor is present in the system, there-upon the typical initialization routine for the Intel486 SX microprocessor will be adequate.

Refer to Appendix B for complete code example.

Following is the interpretation of different combinations of the EM and MP bits.

Table 6.6. EM and MP Bits Interpretations

EM	MP	Interpretation
0	0	Numeric Instructions are Passed to FPU; WAIT Ignores TS
0	1	Numeric Instructions are Passed to FPU; WAIT Tests TS
1	0	Numeric Instructions Trap to Emulator; WAIT Ignores TS
1	1	Numeric Instructions Trap to Emulator; WAIT Tests TS

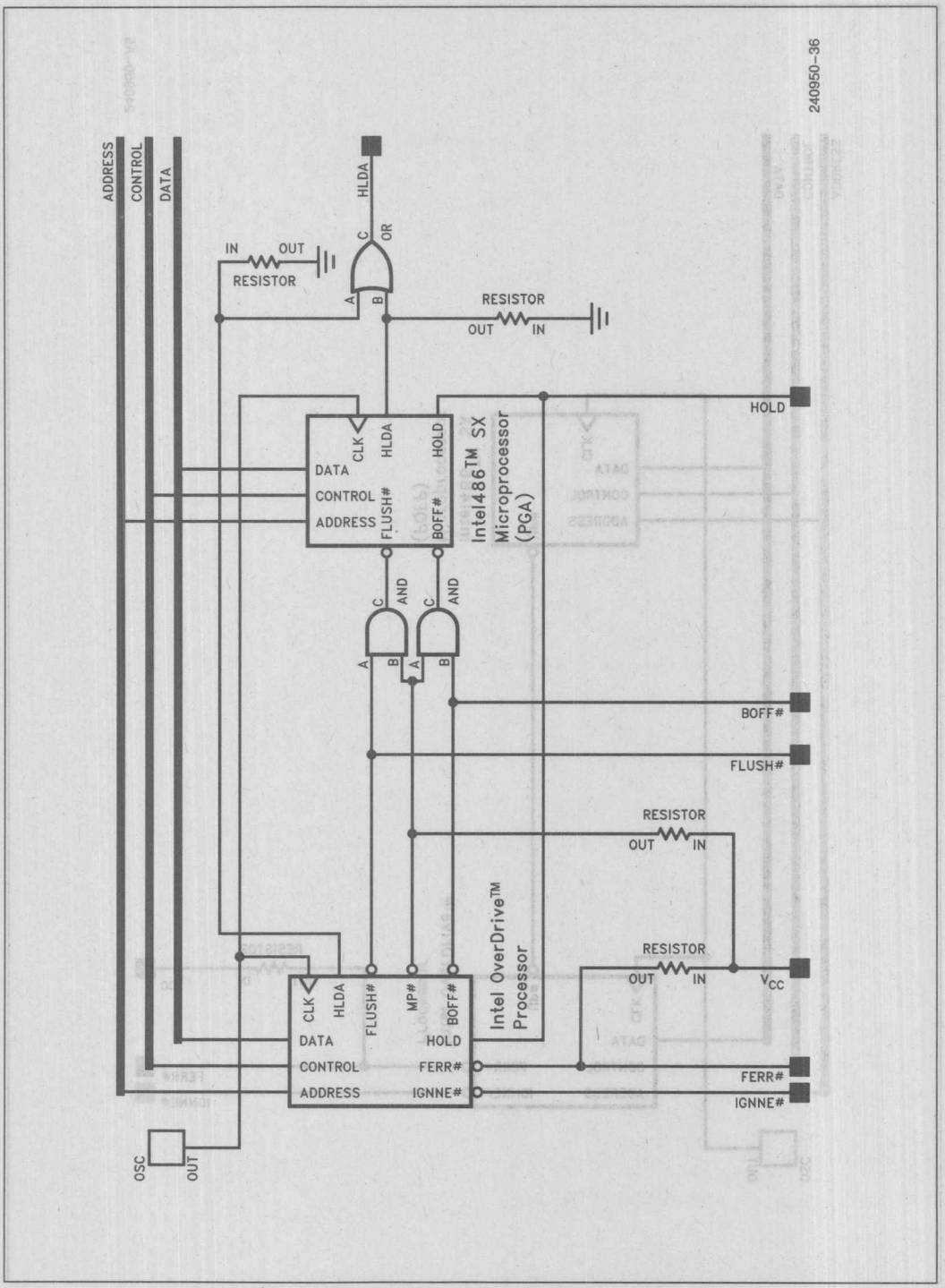


Figure 6.6. Performance-Upgrade Circuit for Intel486™ SX CPU in PGA

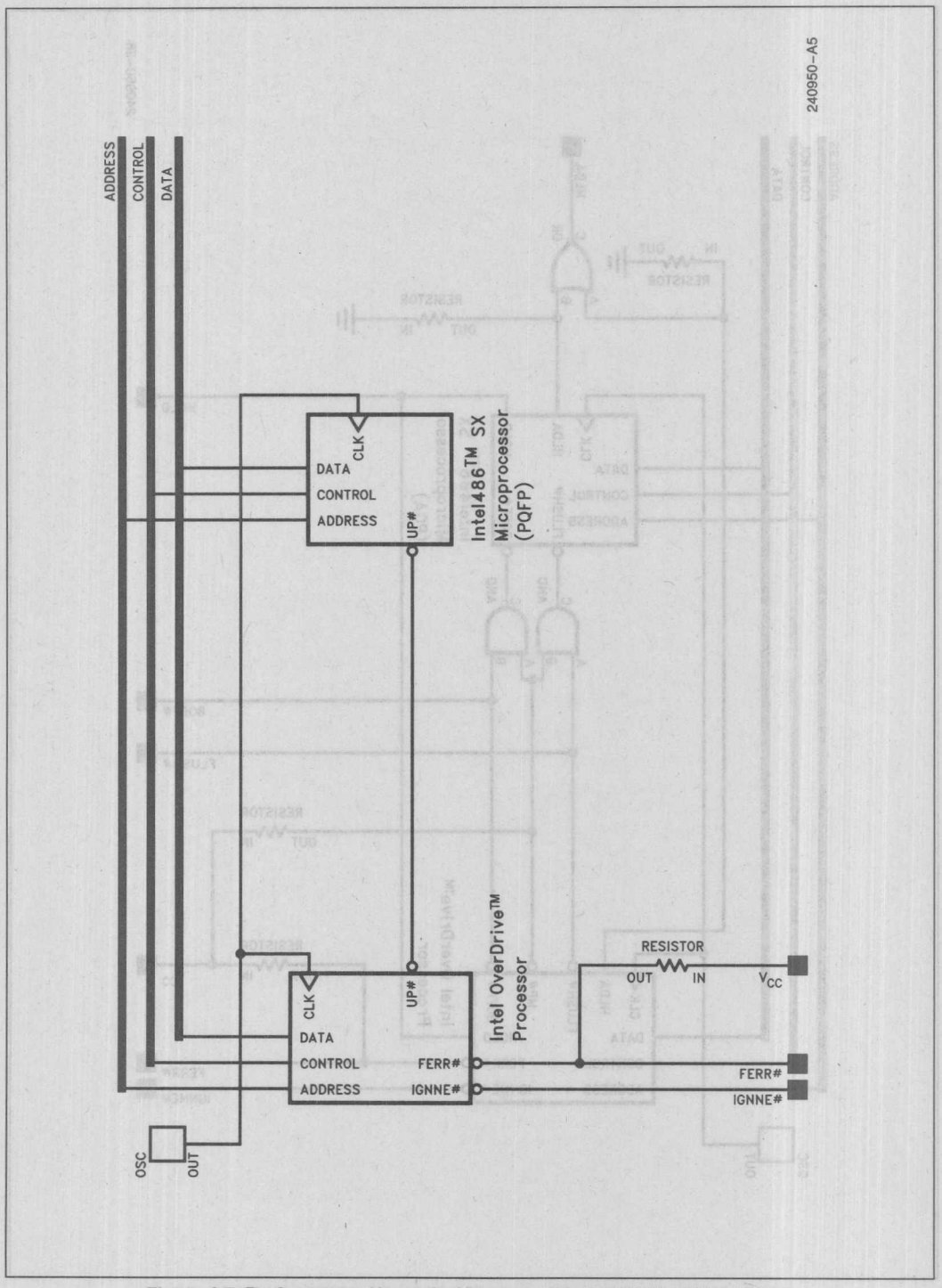


Figure 6.7. Performance-Upgrade Circuit for Intel486™ SX CPU in PQFP

7.0 BUS OPERATION

7.1 Data Transfer Mechanism

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte, word and dword lengths may be transferred without restrictions on physical address alignment. Data may be accessed at any byte boundary but two or three cycles may be required for unaligned data transfers. See Section 7.1.3 Dynamic Bus Sizing and 7.1.6 Operand Alignment.

The Intel486 SX microprocessor/Intel OverDrive Processor address signals are split into two components. High-order address bits are provided by the address lines, A2–A31. The byte enables, BE0#–BE3#, form the low-order address and provide linear selects for the four bytes of the 32-bit address bus.

The byte enable outputs are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 7.1. Byte enable patterns which have a negated byte enable separating two or three asserted byte enables will never occur (see Table 7.5). All other byte enable patterns are possible.

Table 7.1. Byte Enables and Associated Data and Operand Bytes

Byte Enable Signal	Associated Data Bus Signals	
BE0#	D0–D7	(byte 0—least significant)
BE1#	D8–D15	(byte 1)
BE2#	D16–D23	(byte 2)
BE3#	D24–D31	(byte 3—most significant)

Address bits A0 and A1 of the physical operand's base address can be created when necessary. Use of the byte enables to create A0 and A1 is shown in Table 7.2. The byte enables can also be decoded to generate BLE# (byte low enable) and BHE# (byte high enable). These signals are needed to address 16-bit memory systems (see Section 7.1.4 Interfacing with 8- and 16-bit memories).

Table 7.2. Generating A0–A31 from BE0#–BE3# and A2–A31

Intel486™ SX Microprocessor/ OverDrive Processor Address Signals								
A31	..... A2				BE3#	BE2#	BE1#	BE0#
	Physical Base Address							
A31	.....	A2	A1	A0				
A31	.....	A2	0	0	X	X	X	Low
A31	.....	A2	0	1	X	X	Low	High
A31	.....	A2	1	0	X	Low	High	High
A31	.....	A2	1	1	Low	High	High	High

7.1.1 MEMORY AND I/O SPACES

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. Physical memory addresses range from 00000000H to FFFFFFFFH (4 gigabytes). I/O addresses range from 00000000H to 0000FFFFH (64 Kbytes) for programmed I/O. See Figure 7.1.

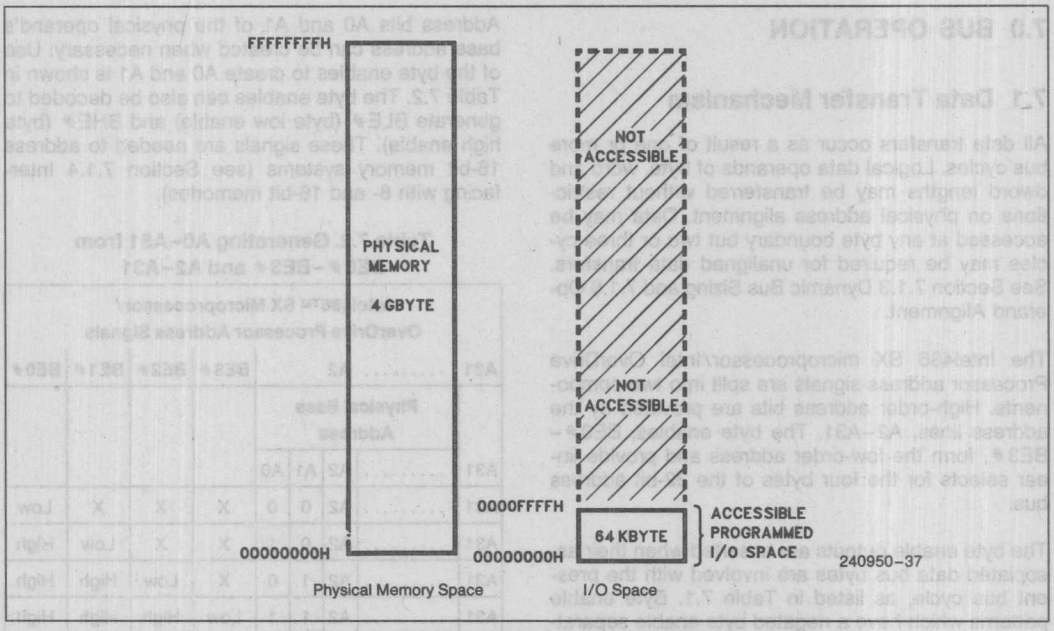


Figure 7.1. Physical Memory and I/O Spaces

7.1.2 MEMORY AND I/O SPACE ORGANIZATION

The Intel486 SX microprocessor/Intel OverDrive Processor datapath to memory and input/output (I/O) spaces can be 32-, 16- or 8-bits wide. The byte enable signals, BE0#–BE3#, allow byte granularity when addressing any memory or I/O structure whether 8, 16 or 32 bits wide.

The Intel486 SX microprocessor/Intel OverDrive Processor includes bus control pins, BS16# and BS8#, which allow direct connection to 16- and 8-bit memories and I/O devices. Cycles to 32-, 16- and 8-bit may occur in any sequence, since the BS8# and BS16# signals are sampled during each bus cycle.

32-bit wide memory and I/O spaces are organized as arrays of physical 4-byte words. Each memory or I/O 4-byte word has four individually addressable bytes at consecutive byte addresses (see Figure 7.2). The lowest addressed byte is associated with data signals D0–D7; the highest-addressed byte with D24–D31. Physical 4-byte words begin at addresses divisible by four.

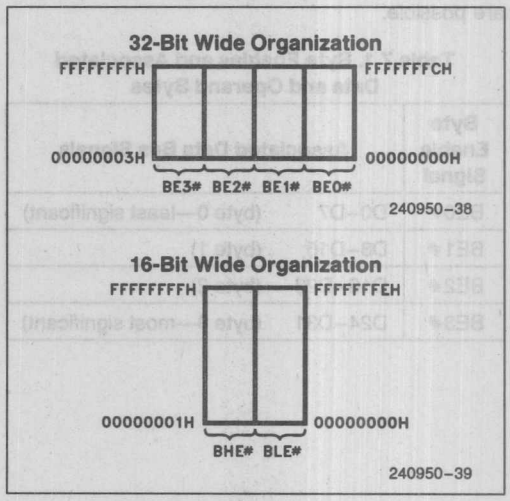


Figure 7.2. Physical Memory and I/O Space Organization

16-bit memories are organized as arrays of physical 2-byte words. Physical 2-byte words begin at addresses divisible by two. The byte enables BE0# – BE3#, must be decoded to A1, BLE# and BHE# to address 16-bit memories (see Section 7.1.4).

To address 8-bit memories, the two low order address bits A0 and A1, must be decoded from BE0# – BE3#. The same logic can be used for 8- and 16-bit memories since the decoding logic for BLE# and A0 are the same (see Section 7.1.4).

7.1.3 DYNAMIC DATA BUS SIZING

Dynamic data bus sizing is a feature allowing processor connection to 32-, 16- or 8-bit buses for memory or I/O. The Intel486 SX microprocessor/Intel OverDrive Processor may connect to all three bus sizes. Transfers to or from 32-, 16- or 8-bit devices are supported by dynamically determining the bus width during each bus cycle. Address decoding circuitry may assert BS16# for 16-bit devices, or BS8# for 8-bit devices during each bus cycle. BS8# and BS16# must be negated when addressing 32-bit devices. An 8-bit bus width is selected if both BS16# and BS8# are asserted.

BS16# and BS8# force the Intel486 SX microprocessor/Intel OverDrive Processor to run additional bus cycles to complete requests larger than 16- or 8 bits. A 32-bit transfer will be converted into two 16-bit transfers (or 3 transfers if the data is misaligned) when BS16# is asserted. Asserting BS8# will convert a 32-bit transfer into four 8-bit transfers.

Extra cycles forced by BS16# or BS8# should be viewed as independent bus cycles. BS16# or BS8# must be driven active during each of the extra cycles unless the addressed device has the ability to change the number of bytes it can return between cycles.

The Intel486 SX microprocessor/Intel OverDrive Processor will drive the byte enables appropriately during extra cycles forced by BS8# and BS16#. A2–A31 will not change if accesses are to a 32-bit aligned area. Table 7.3 shows the set of byte enables that will be generated on the next cycle for each of the valid possibilities of the byte enables on the current cycle.

The dynamic bus sizing feature of the Intel486 SX microprocessor/Intel OverDrive Processor is significantly different than that of the Intel386 microprocessor. Unlike the Intel386 microprocessor, the Intel486 SX microprocessor/Intel OverDrive Processor requires that data bytes be driven on the addressed data pins. The simplest example of this function is a 32-bit aligned, BS16# read. When the Intel486 SX microprocessor/Intel OverDrive Processor reads the two high order bytes, they must be driven on the data bus pins D16–D31. The Intel486 SX microprocessor/Intel OverDrive Processor expects the two low order bytes on D0–D15. The Intel386 microprocessor expects both the high and low order bytes on D0–D15. The Intel386 microprocessor always reads or writes data on the lower 16 bits of the data bus when BS16# is asserted.

The external system must contain buffers to enable the Intel486 SX microprocessor/Intel OverDrive Processor to read and write data on the appropriate data bus pins. Table 7.4 shows the data bus lines where the Intel486 SX microprocessor/Intel OverDrive Processor expects data to be returned for each valid combination of byte enables and bus sizing options.

Valid data will only be driven onto data bus pins corresponding to active byte enables during write cycles. Other pins in the data bus will be driven but they will not contain valid data. Unlike the Intel386 microprocessor, the Intel486 SX microprocessor/Intel OverDrive Processor will not duplicate write data onto parts of the data bus for which the corresponding byte enable is negated.

Table 7.3. Next Byte Enable Values for BSn# Cycles

Current				Next with BS8#				Next with BS16#			
BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#
1	1	1	0	n	n	n	n	n	n	n	n
1	1	0	0	1	1	0	1	n	n	n	n
1	0	0	0	1	0	0	1	1	0	1	1
0	0	0	0	0	0	0	1	0	0	1	1
1	1	0	1	n	n	n	n	n	n	n	n
1	0	0	1	1	0	1	1	1	0	1	1
0	0	0	1	0	0	1	1	0	0	1	1
1	0	1	1	n	n	n	n	n	n	n	n
0	0	1	1	0	1	1	1	n	n	n	n
0	1	1	1	n	n	n	n	n	n	n	n

"n" means that another bus cycle will not be required to satisfy the request.

Table 7.4. Data Pins Read with Different Bus Sizes

BE3#	BE2#	BE1#	BE0#	w/o BS8#/BS16#	w BS8#	W BS16#
1	1	1	0	D7–D0	D7–D0	D7–D0
1	1	0	0	D15–D0	D7–D0	D15–D0
1	0	0	0	D23–D0	D7–D0	D15–D0
0	0	0	0	D31–D0	D7–D0	D15–D0
1	1	0	1	D15–D8	D15–D8	D15–D8
1	0	0	1	D23–D8	D15–D8	D15–D8
0	0	0	1	D31–D8	D15–D8	D15–D8
1	0	1	1	D23–D16	D23–D16	D23–D16
0	0	1	1	D31–D16	D23–D16	D31–D16
0	1	1	1	D31–D24	D31–D24	D31–D24

7.1.4 INTERFACING WITH 8-, 16- AND 32-BIT MEMORIES

In 32-bit physical memories such as Figure 7.3, each 4-byte word begins at a byte address that is a multiple of four. A2–A31 are used as a 4-byte word select. BE0#–BE3# select individual bytes within the 4-byte word. BS8# and BS16# are negated for all bus cycles involving the 32-bit array.

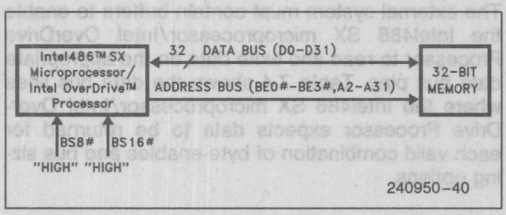


Figure 7.3. Intel486™ SX Microprocessor/ Intel OverDrive Processor with 32-Bit Memory

16- and 8-bit memories require external byte swapping logic for routing data to the appropriate data lines and logic for generating BHE#, BLE# and A1. In systems where mixed memory widths are used, extra address decoding logic is necessary to assert BS16# or BS8#.

Figure 7.4 shows the Intel486 SX microprocessor/ Intel OverDrive Processor address bus interface to 32-, 16- and 8-bit memories. To address 16-bit memories the byte enables must be decoded to produce A1, BHE# and BLE# (A0). For 8-bit wide memories the byte enables must be decoded to produce A0 and A1. The same byte select logic can be used in 16- and 8-bit systems since BLE# is exactly the same as A0 (see Table 7.5).

BE0#–BE3# can be decoded as shown in Table 7.5 to generate A1, BHE# and BLE#. The byte select logic necessary to generate BHE# and BLE# is shown in Figure 7.5.

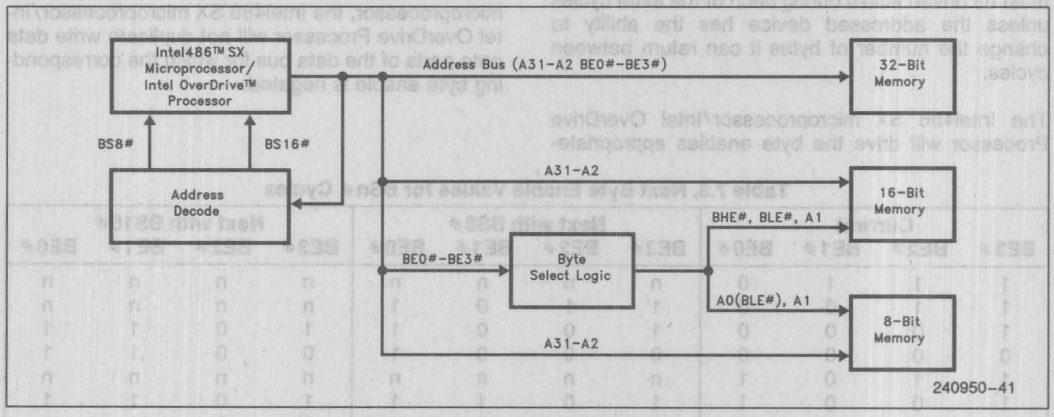


Figure 7.4. Addressing 16- and 8-Bit Memories

Table 7.5. Generating A1, BHE# and BLE# for Addressing 16-Bit Devices

Intel486™ SX Microprocessor/ Intel OverDrive Processor				8-, 16-Bit Bus Signals			Comments
BE3#	BE2#	BE1#	BE0#	A1	BHE#	BLE# (A0)	
H*	H*	H*	H*	x	x	x	x—no active bytes
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	
H*	L*	H*	L*	x	x	x	x—not contiguous bytes
H	L	L	H	L	L	H	
H	L	L	L	L	L	L	
L	H	H	H	H	L	H	
L*	H*	H*	L*	x	x	x	x—not contiguous bytes
L*	H*	L*	H*	x	x	x	x—not contiguous bytes
L*	H*	L*	L*	x	x	x	x—not contiguous bytes
L	L	H	H	H	L	L	
L*	L*	H*	L*	x	x	x	x—not contiguous bytes
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

BLE# asserted when D0–D7 of 16-bit bus is active.
BHE# asserted when D8–D15 of 16-bit bus is active.
A1 low for all even words; A1 high for all odd words.

Key:
x = don't care
H = high voltage level
L = low voltage level
* = a non-occurring pattern of Byte Enables; either none are asserted, or the pattern has Byte Enables asserted for non-contiguous bytes

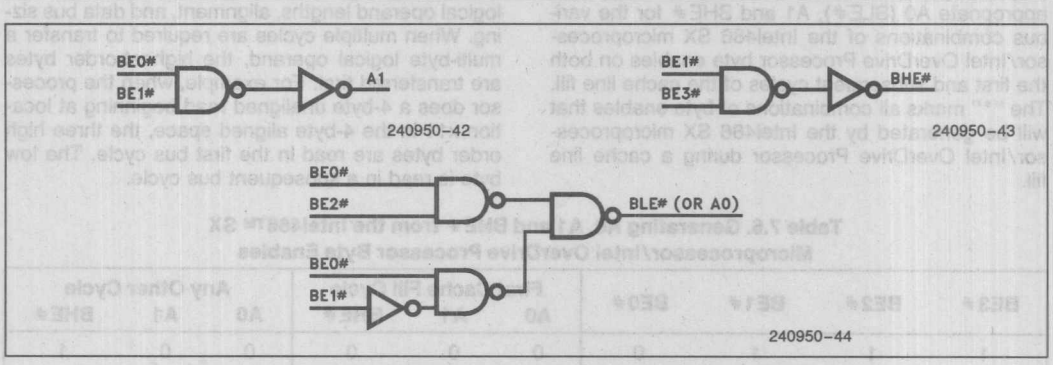


Figure 7.5. Logic to Generate A1, BHE# and BLE# for 16-Bit Busses

Combinations of BE0#–BE3# which never occur are those in which two or three asserted byte enables are separated by one or more negated byte enables. These combinations are "don't care" conditions in the decoder. A decoder can use the non-occurring BE0#–BE3# combinations to its best advantage.

Figure 7.6 shows an Intel486 SX microprocessor/Intel OverDrive Processor data bus interface to 16- and 8-bit wide memories. External byte swapping logic is needed on the data lines so that data is supplied to, and received from the Intel486 SX microprocessor/Intel OverDrive Processor on the correct data pins (see Table 7.4).

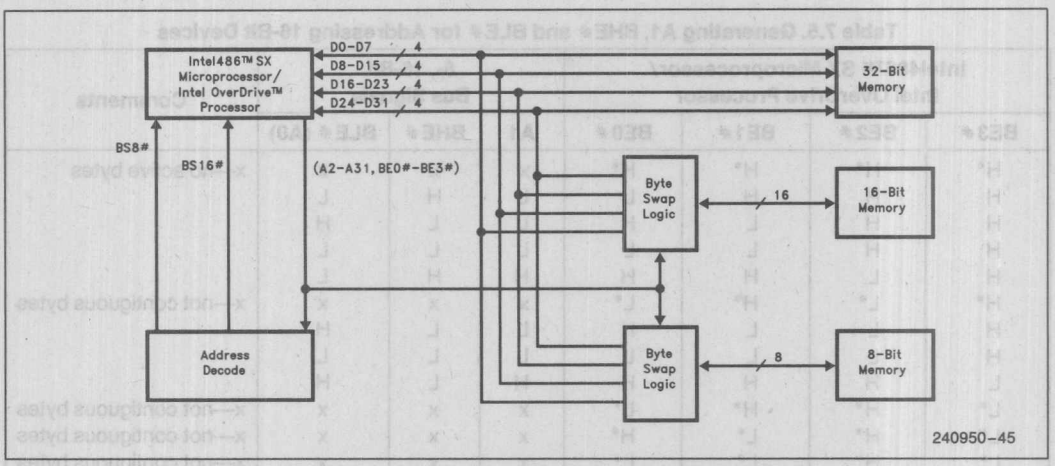


Figure 7.6. Data Bus Interface to 16- and 8-Bit Memories

7.1.5 DYNAMIC BUS SIZING DURING CACHE LINE FILLS

BS8# and BS16# can be driven during cache line fills. The Intel486 SX microprocessor/Intel OverDrive Processor will generate enough 8- or 16-bit cycles to fill the cache line. This can be up to 16 8-bit cycles.

The external system should assume that all byte enables are active for the first cycle of a cache line fill. The Intel486 SX microprocessor/Intel OverDrive Processor will generate proper byte enables for subsequent cycles in the line fill. Table 7.6 shows the appropriate A0 (BLE#), A1 and BHE# for the various combinations of the Intel486 SX microprocessor/Intel OverDrive Processor byte enables on both the first and subsequent cycles of the cache line fill. The "*" marks all combinations of byte enables that will be generated by the Intel486 SX microprocessor/Intel OverDrive Processor during a cache line fill.

7.1.6 OPERAND ALIGNMENT

Physical 4-byte words begin at addresses that are multiples of four. It is possible to transfer a logical operand that spans more than one physical 4-byte word of memory or I/O at the expense of extra cycles. Examples are 4-byte operands beginning at addresses that are not evenly divisible by 4, or 2-byte words split between two physical 4-byte words. These are referred to as unaligned transfers.

Operand alignment and data bus size dictate when multiple bus cycles are required. Table 7.7 describes the transfer cycles generated for all combinations of logical operand lengths, alignment, and data bus sizing. When multiple cycles are required to transfer a multi-byte logical operand, the highest-order bytes are transferred first. For example, when the processor does a 4-byte unaligned read beginning at location x11 in the 4-byte aligned space, the three high order bytes are read in the first bus cycle. The low byte is read in a subsequent bus cycle.

Table 7.6. Generating A0, A1 and BHE# from the Intel486™ SX Microprocessor/Intel OverDrive Processor Byte Enables

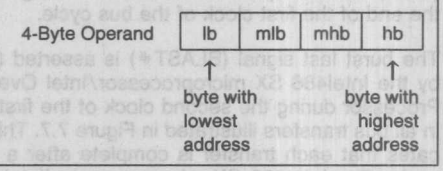
BE3#	BE2#	BE1#	BE0#	First Cache Fill Cycle			Any Other Cycle		
				A0	A1	BHE#	A0	A1	BHE#
1	1	1	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
*0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	1	0	0
1	0	0	1	0	0	0	1	0	0
*0	0	0	1	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	1
*0	0	1	1	0	0	0	0	1	0
*0	1	1	1	0	0	0	1	1	0

Table 7.7. Transfer Bus Cycles for Bytes, Words and Dwords

Byte-Length of Logical Operand	1	2				4			
Physical Byte Address in Memory (Low Order Bits)	xx	00	01	10	11	00	01	10	11
Transfer Cycles over 32-Bit Bus	b	w	w	w	hb lb	d	hb l3	hw lw	h3 lb
Transfer Cycles over 16-Bit Data Bus ▨ = BS16# Asserted	b	w	lb hb	w	hb lb	lw hw	hb lb mw	hw lw	mw hb lb
Transfer Cycles over 8-Bit Data Bus ▨ = BS8# Asserted	b	lb hb	lb hb	lb hb	hb lb	lb mlb mhb hb	hb lb mlb mhb	mhb hb lb mlb	mlb mhb hb lb

240950-99

KEY:
b = byte transfer
w = 2-byte transfer
3 = 3-byte transfer
d = 4-byte transfer
h = high-order portion
l = low-order portion
m = mid-order portion



The function of unaligned transfers with dynamic bus sizing is not obvious. When the external systems asserts BS16# or BS8# forcing extra cycles, low-order bytes or words are transferred first (opposite to the example above). When the Intel486 SX microprocessor/Intel OverDrive Processor requests a 4-byte read and the external system asserts BS16#, the lower 2 bytes are read first followed by the upper 2 bytes.

In the unaligned transfer described above, the processor requested three bytes on the first cycle. If the external system asserted BS16# during this 3-byte transfer, the lower word is transferred first followed by the upper byte. In the final cycle the lower byte of the 4-byte operand is transferred as in the 32-bit example above.

7.2 Bus Functional Description

The Intel486 SX microprocessor/Intel OverDrive Processor supports a wide variety of bus transfers to meet the needs of high performance systems. Bus transfers can be single cycle or multiple cycle, burst or non-burst, cacheable or non-cacheable, 8-, 16- or 32-bit, and pseudo-locked. To support multiprocessing systems there are cache invalidation cycles and locked cycles.

This section begins with basic non-cacheable non-burst single cycle transfers. It moves on to multiple cycle transfers and introduces the burst mode. Cacheability is introduced in Section 7.2.3. The remaining sections describe locked, pseudo-locked, invalidate, bus hold and interrupt cycles.

Bus cycles and data cycles are discussed in this section. A bus cycle is at least two clocks long and begins with ADS# active in the first clock and ready active in the last clock. Data is transferred to or from the Intel486 SX microprocessor/Intel OverDrive Processor during a data cycle. A bus cycle contains one or more data cycles.

Refer to Section 7.2.13 for a description of the bus states shown in the timing diagrams.

7.2.1 NON-CACHEABLE NON-BURST SINGLE CYCLE

7.2.1.1 No Wait States

The fastest non-burst bus cycle that the Intel486 SX microprocessor/Intel OverDrive Processor supports is two clocks long. These cycles are called 2-2 cycles because reads and writes take two cycles each. The first 2 refers to reads and the second to writes.

For example, if a wait state needs to be added to a write, the cycle would be called 2-3.

Basic two clock read and write cycles are shown in Figure 7.7. The Intel486 SX microprocessor/Intel OverDrive Processor initiates a cycle by asserting the address status signal (ADS#) at the rising edge of the first clock. The ADS# output indicates that a valid bus cycle definition and address is available on the cycle definition lines and address bus.

The non-burst ready input (RDY#) is returned by the external system in the second clock. RDY# indicates that the external system has presented valid data on the data pins in response to a read or the external system has accepted data in response to a write.

The Intel486 SX microprocessor/Intel OverDrive Processor samples RDY# at the end of the second clock. The cycle is complete if RDY# is active (LOW) when sampled. Note that RDY# is ignored at the end of the first clock of the bus cycle.

The burst last signal (BLAST#) is asserted (LOW) by the Intel486 SX microprocessor/Intel OverDrive Processor during the second clock of the first cycle in all bus transfers illustrated in Figure 7.7. This indicates that each transfer is complete after a single cycle. The Intel486 SX microprocessor/Intel OverDrive Processor asserts BLAST# in the last cycle of a bus transfer.

The timing of the parity check output (PCHK#) is shown in Figure 7.7. The Intel486 SX microprocessor/Intel OverDrive Processor drives the PCHK# output one clock after ready terminates a read cycle. PCHK# indicates the parity status for the data sampled at the end of the previous clock. The PCHK# signal can be used by the external system. The Intel486 SX microprocessor/Intel OverDrive Processor does nothing in response to the PCHK# output.

7.2.1.2 Inserting Wait States

The external system can insert wait states into the basic 2-2 cycle by driving RDY# inactive at the end of the second clock. RDY# must be driven inactive to insert a wait state. Figure 7.8 illustrates a simple non-burst, non-cacheable signal with one wait state added. Any number of wait states can be added to an Intel486 SX microprocessor/Intel OverDrive Processor bus cycle by maintaining RDY# inactive.

The burst ready input (BRDY#) must be driven inactive on all clock edges where RDY# is driven inactive for proper operation of these simple non-burst cycles.

7.2.2 MULTIPLE AND BURST CYCLE BUS TRANSFERS

Multiple cycle bus transfers can be caused by internal requests from the Intel486 SX microprocessor/Intel OverDrive Processor or by the external memory system. An internal request for a 128-bit pre-fetch must take more than one cycle. Internal requests for unaligned data may also require multiple bus cycles. A cache line fill requires multiple cycles to complete.

> An internal request by the Intel OverDrive Processor for a 64-bit floating point load must take more than one internal cycle.

The external system can cause a multiple cycle transfer when it can only supply 8 or 16 bits per cycle.

Only multiple cycle transfers caused by internal requests are considered in this section. Cacheable cycles and 8- and 16-bit transfers are covered in Sections 7.2.3 and 7.2.5.

7.2.2.1 Burst Cycles

The Intel486 SX microprocessor/Intel OverDrive Processor can accept burst cycles for any bus requests that require more than a single data cycle. During burst cycles, a new data item is strobed into the Intel486 SX microprocessor/Intel OverDrive Processor every clock rather than every other clock as in non-burst cycles. The fastest burst cycle requires 2 clocks for the first data item with subsequent data items returned every clock.

The Intel486 SX microprocessor/Intel OverDrive Processor is capable of bursting a maximum of 32 bits during a write. Burst writes can only occur if BS8# or BS16# is asserted. For example, the Intel486 SX microprocessor/Intel OverDrive Processor can burst write four 8-bit operands or two 16-bit operands in a single burst cycle. But the Intel486 SX microprocessor/Intel OverDrive Processor cannot burst multiple 32-bit writes in a single burst cycle.

Burst cycles begin with the Intel486 SX microprocessor/Intel OverDrive Processor driving out an address and asserting ADS# in the same manner as non-burst cycles. The Intel486 SX microprocessor/Intel OverDrive Processor indicates that it is willing to perform a burst cycle by holding the burst last signal (BLAST#) inactive in the second clock of the cycle. The external system indicates its willingness to do a burst cycle by returning the burst ready signal (BRDY#) active.

The addresses of the data items in a burst cycle will all fall within the same 16-byte aligned area (corresponding to an internal Intel486 SX microprocessor/Intel OverDrive Processor cache line). A 16-byte aligned area begins at location XXXXXXX0 and ends at location XXXXXXXF. During a burst cycle, only BE0–3#, A_2, and A_3 may change. A_4–A_{31}, M/IO#, D/C#, and W/R# will remain stable throughout a burst. Given the first address in a burst, external hardware can easily calculate the address of subsequent transfers in advance. An external memory system can be designed to quickly fill the Intel486 SX microprocessor/Intel OverDrive Processor internal cache lines.

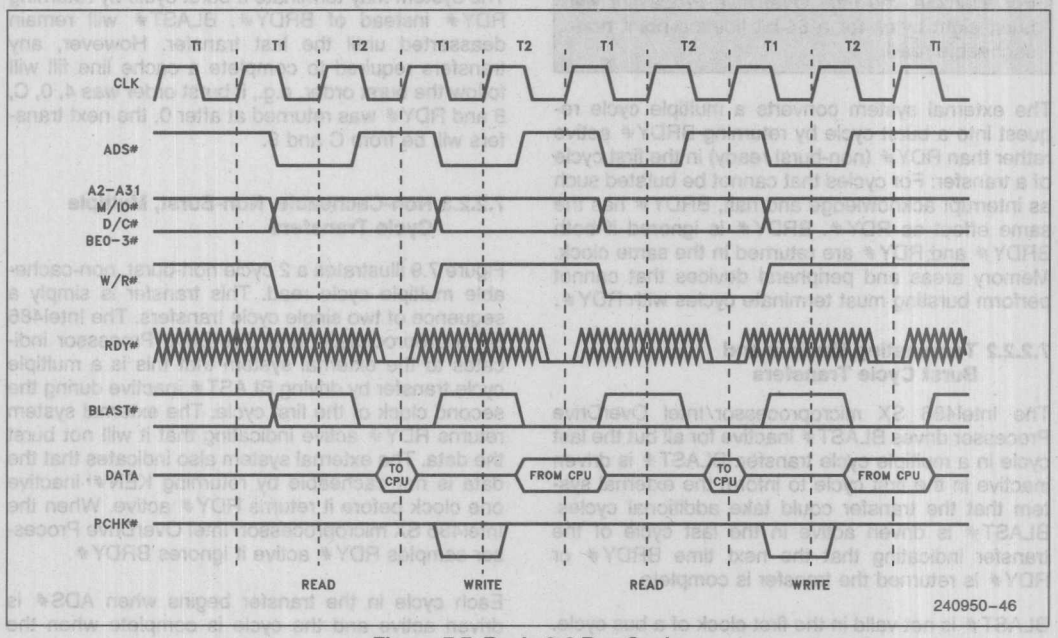


Figure 7.7. Basic 2-2 Bus Cycle

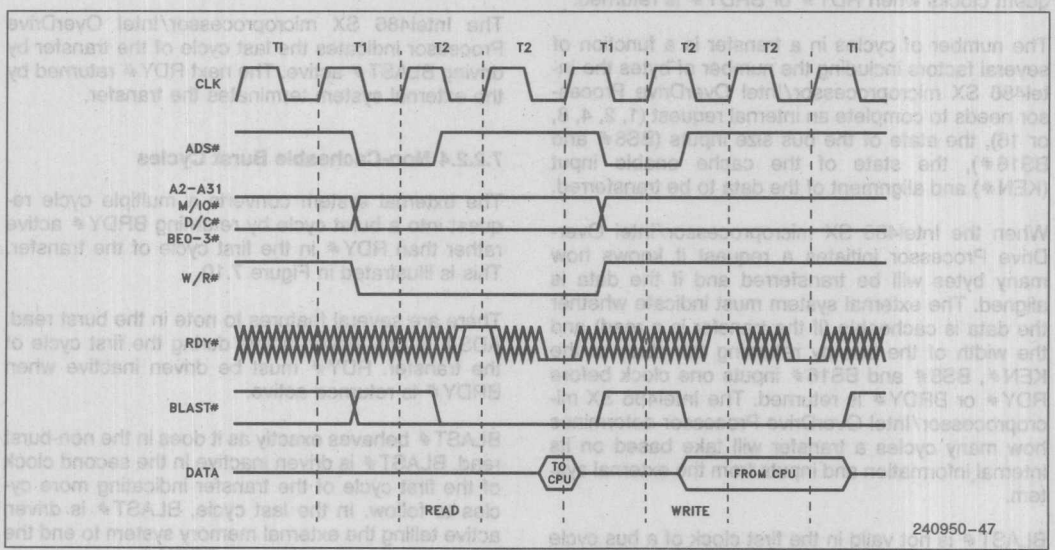


Figure 7.8. Basic 3-3 Bus Cycle

Burst cycles are not limited to cache line fills. Any multiple cycle read request by the Intel486 SX microprocessor/Intel OverDrive Processor can be converted into a burst cycle. The Intel486 SX microprocessor/Intel OverDrive Processor will only burst the number of bytes needed to complete a transfer.

> For example, the Intel OverDrive Processor will burst eight bytes for a 64-bit floating point non-cacheable read.

The external system converts a multiple cycle request into a burst cycle by returning BRDY# active rather than RDY# (non-burst ready) in the first cycle of a transfer. For cycles that cannot be bursted such as interrupt acknowledge and halt, BRDY# has the same effect as RDY#. BRDY# is ignored if both BRDY# and RDY# are returned in the same clock. Memory areas and peripheral devices that cannot perform bursting must terminate cycles with RDY#.

7.2.2.2 Terminating Multiple and Burst Cycle Transfers

The Intel486 SX microprocessor/Intel OverDrive Processor drives BLAST# inactive for all but the last cycle in a multiple cycle transfer. BLAST# is driven inactive in the first cycle to inform the external system that the transfer could take additional cycles. BLAST# is driven active in the last cycle of the transfer indicating that the next time BRDY# or RDY# is returned the transfer is complete.

BLAST# is not valid in the first clock of a bus cycle. It should be sampled only in the second and subsequent clocks when RDY# or BRDY# is returned.

The number of cycles in a transfer is a function of several factors including the number of bytes the Intel486 SX microprocessor/Intel OverDrive Processor needs to complete an internal request (1, 2, 4, 8, or 16), the state of the bus size inputs (BS8# and BS16#), the state of the cache enable input (KEN#) and alignment of the data to be transferred.

When the Intel486 SX microprocessor/Intel OverDrive Processor initiates a request it knows how many bytes will be transferred and if the data is aligned. The external system must indicate whether the data is cacheable (if the transfer is a read) and the width of the bus by returning the state of the KEN#, BS8# and BS16# inputs one clock before RDY# or BRDY# is returned. The Intel486 SX microprocessor/Intel OverDrive Processor determines how many cycles a transfer will take based on its internal information and inputs from the external system.

BLAST# is not valid in the first clock of a bus cycle because the Intel486 SX microprocessor/Intel OverDrive Processor cannot determine the number of cycles a transfer will take until the external system returns KEN#, BS8# and BS16#. BLAST# should only be sampled in the second and subsequent clocks of a cycle when the external system returns RDY# or BRDY#.

The system may terminate a burst cycle by returning RDY# instead of BRDY#. BLAST# will remain deasserted until the last transfer. However, any transfers required to complete a cache line fill will follow the burst order, e.g., if burst order was 4, 0, C, 8 and RDY# was returned at after 0, the next transfers will be from C and 8.

7.2.2.3 Non-Cacheable, Non-Burst, Multiple Cycle Transfers

Figure 7.9 illustrates a 2 cycle non-burst, non-cacheable multiple cycle read. This transfer is simply a sequence of two single cycle transfers. The Intel486 SX microprocessor/Intel OverDrive Processor indicates to the external system that this is a multiple cycle transfer by driving BLAST# inactive during the second clock of the first cycle. The external system returns RDY# active indicating that it will not burst the data. The external system also indicates that the data is not cacheable by returning KEN# inactive one clock before it returns RDY# active. When the Intel486 SX microprocessor/Intel OverDrive Processor samples RDY# active it ignores BRDY#.

Each cycle in the transfer begins when ADS# is driven active and the cycle is complete when the external system returns RDY# active.

The Intel486 SX microprocessor/Intel OverDrive Processor indicates the last cycle of the transfer by driving BLAST# active. The next RDY# returned by the external system terminates the transfer.

7.2.2.4 Non-Cacheable Burst Cycles

The external system converts a multiple cycle request into a burst cycle by returning BRDY# active rather than RDY# in the first cycle of the transfer. This is illustrated in Figure 7.10.

There are several features to note in the burst read. ADS# is only driven active during the first cycle of the transfer. RDY# must be driven inactive when BRDY# is returned active.

BLAST# behaves exactly as it does in the non-burst read. BLAST# is driven inactive in the second clock of the first cycle of the transfer indicating more cycles to follow. In the last cycle, BLAST# is driven active telling the external memory system to end the burst after returning the next BRDY#.

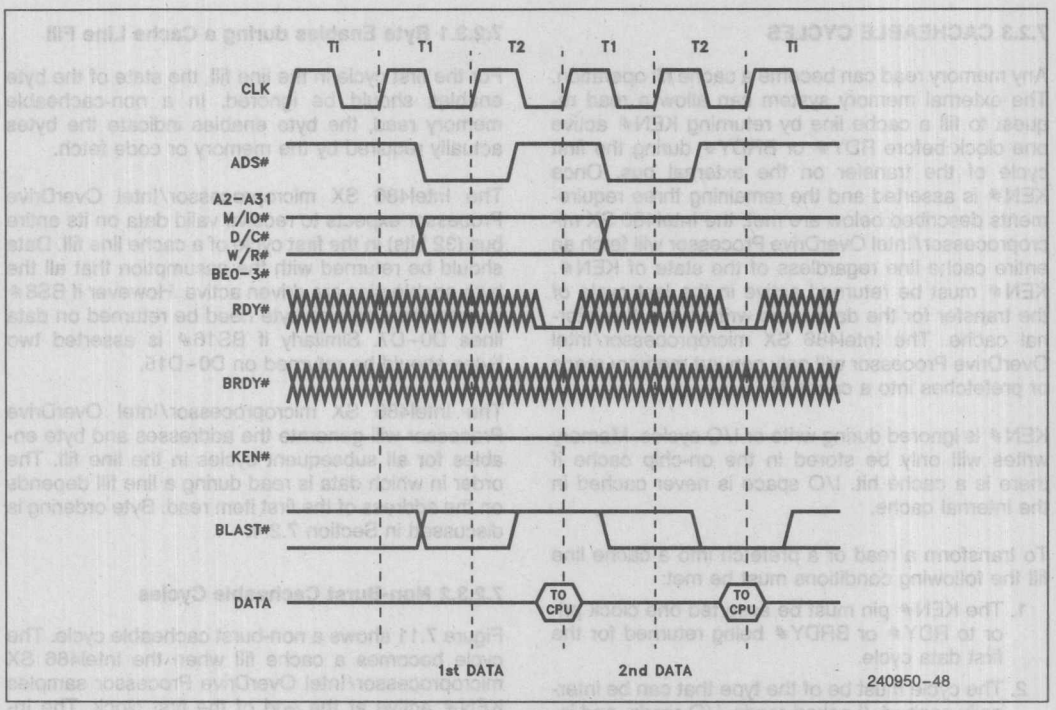


Figure 7.9. Non-Cacheable, Non-Burst, Multiple Cycle Transfers

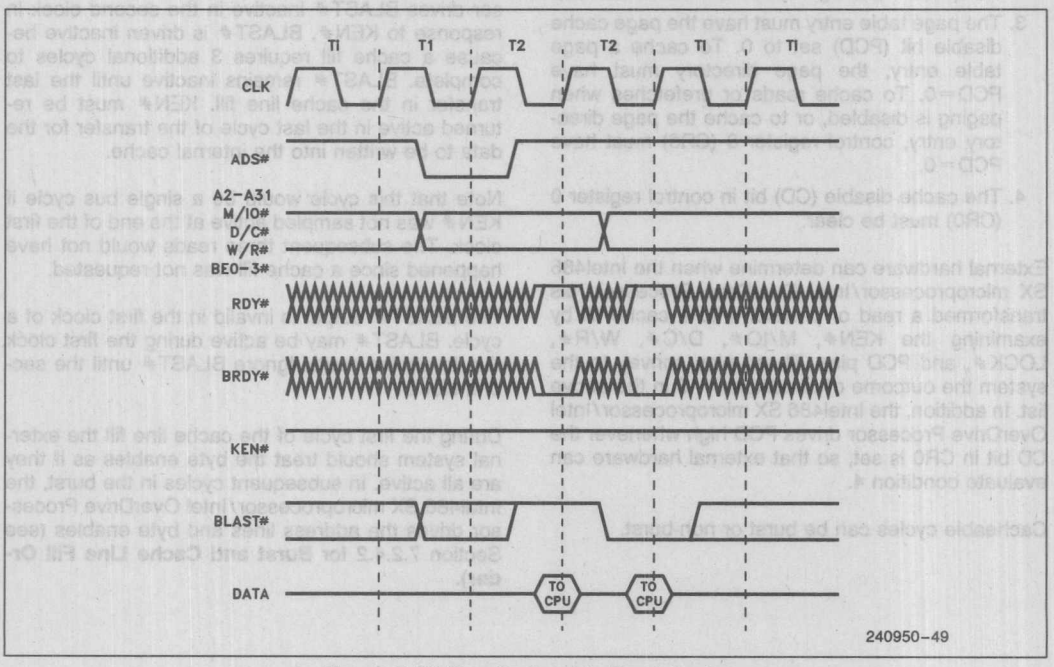


Figure 7.10. Non-Cacheable Burst Cycle

7.2.3 CACHEABLE CYCLES

Any memory read can become a cache fill operation. The external memory system can allow a read request to fill a cache line by returning KEN# active one clock before RDY# or BRDY# during the first cycle of the transfer on the external bus. Once KEN# is asserted and the remaining three requirements described below are met, the Intel486 SX microprocessor/Intel OverDrive Processor will fetch an entire cache line regardless of the state of KEN#. KEN# must be returned active in the last cycle of the transfer for the data to be written into the internal cache. The Intel486 SX microprocessor/Intel OverDrive Processor will only convert memory reads or prefetches into a cache fill.

KEN# is ignored during write or I/O cycles. Memory writes will only be stored in the on-chip cache if there is a cache hit. I/O space is never cached in the internal cache.

To transform a read or a prefetch into a cache line fill the following conditions must be met:

1. The KEN# pin must be asserted one clock prior to RDY# or BRDY# being returned for the first data cycle.
2. The cycle must be of the type that can be internally cached. (Locked reads, I/O reads, and interrupt acknowledge cycles are never cached).
3. The page table entry must have the page cache disable bit (PCD) set to 0. To cache a page table entry, the page directory must have PCD=0. To cache reads or prefetches when paging is disabled, or to cache the page directory entry, control register 3 (CR3) must have PCD=0.
4. The cache disable (CD) bit in control register 0 (CR0) must be clear.

External hardware can determine when the Intel486 SX microprocessor/Intel OverDrive Processor has transformed a read or prefetch into a cache fill by examining the KEN#, M/IO#, D/C#, W/R#, LOCK#, and PCD pins. These pins convey to the system the outcome of conditions 1–3 in the above list. In addition, the Intel486 SX microprocessor/Intel OverDrive Processor drives PCD high whenever the CD bit in CR0 is set, so that external hardware can evaluate condition 4.

Cacheable cycles can be burst or non-burst.

7.2.3.1 Byte Enables during a Cache Line Fill

For the first cycle in the line fill, the state of the byte enables should be ignored. In a non-cacheable memory read, the byte enables indicate the bytes actually required by the memory or code fetch.

The Intel486 SX microprocessor/Intel OverDrive Processor expects to receive valid data on its entire bus (32 bits) in the first cycle of a cache line fill. Data should be returned with the assumption that all the byte enable pins are driven active. However if BS8# is asserted only one byte need be returned on data lines D0–D7. Similarly if BS16# is asserted two bytes should be returned on D0–D15.

The Intel486 SX microprocessor/Intel OverDrive Processor will generate the addresses and byte enables for all subsequent cycles in the line fill. The order in which data is read during a line fill depends on the address of the first item read. Byte ordering is discussed in Section 7.2.4.

7.2.3.2 Non-Burst Cacheable Cycles

Figure 7.11 shows a non-burst cacheable cycle. The cycle becomes a cache fill when the Intel486 SX microprocessor/Intel OverDrive Processor samples KEN# active at the end of the first clock. The Intel486 SX microprocessor/Intel OverDrive Processor drives BLAST# inactive in the second clock in response to KEN#. BLAST# is driven inactive because a cache fill requires 3 additional cycles to complete. BLAST# remains inactive until the last transfer in the cache line fill. KEN# must be returned active in the last cycle of the transfer for the data to be written into the internal cache.

Note that this cycle would be a single bus cycle if KEN# was not sampled active at the end of the first clock. The subsequent three reads would not have happened since a cache fill was not requested.

The BLAST# output is invalid in the first clock of a cycle. BLAST# may be active during the first clock due to earlier inputs. Ignore BLAST# until the second clock.

During the first cycle of the cache line fill the external system should treat the byte enables as if they are all active. In subsequent cycles in the burst, the Intel486 SX microprocessor/Intel OverDrive Processor drives the address lines and byte enables (see Section 7.2.4.2 for **Burst and Cache Line Fill Order**).

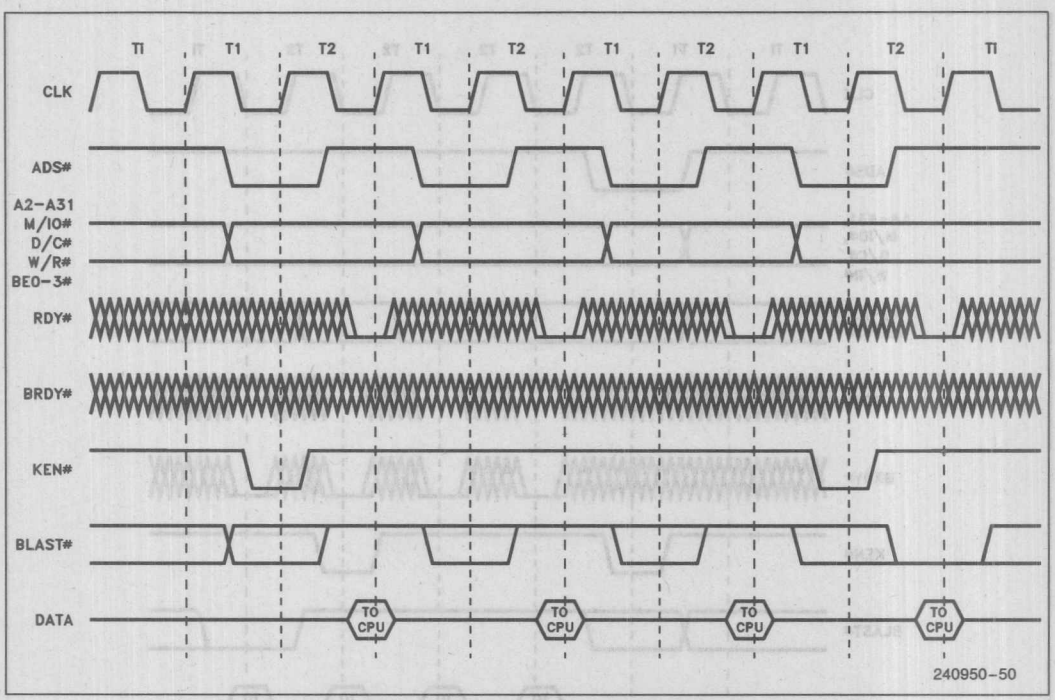


Figure 7.11. Non-Burst, Cacheable Cycles

7.2.3.3 Burst Cacheable Cycles

Figure 7.12 illustrates a burst mode cache fill. As in Figure 7.11, the transfer becomes a cache line fill when the external system returns KEN# active at the end of the first clock in the cycle.

The external system informs the Intel486 SX microprocessor/Intel OverDrive Processor that it will burst the line in by driving BRDY# active at the end of the first cycle in the transfer.

Note that during a burst cycle ADS# is only driven with the first address.

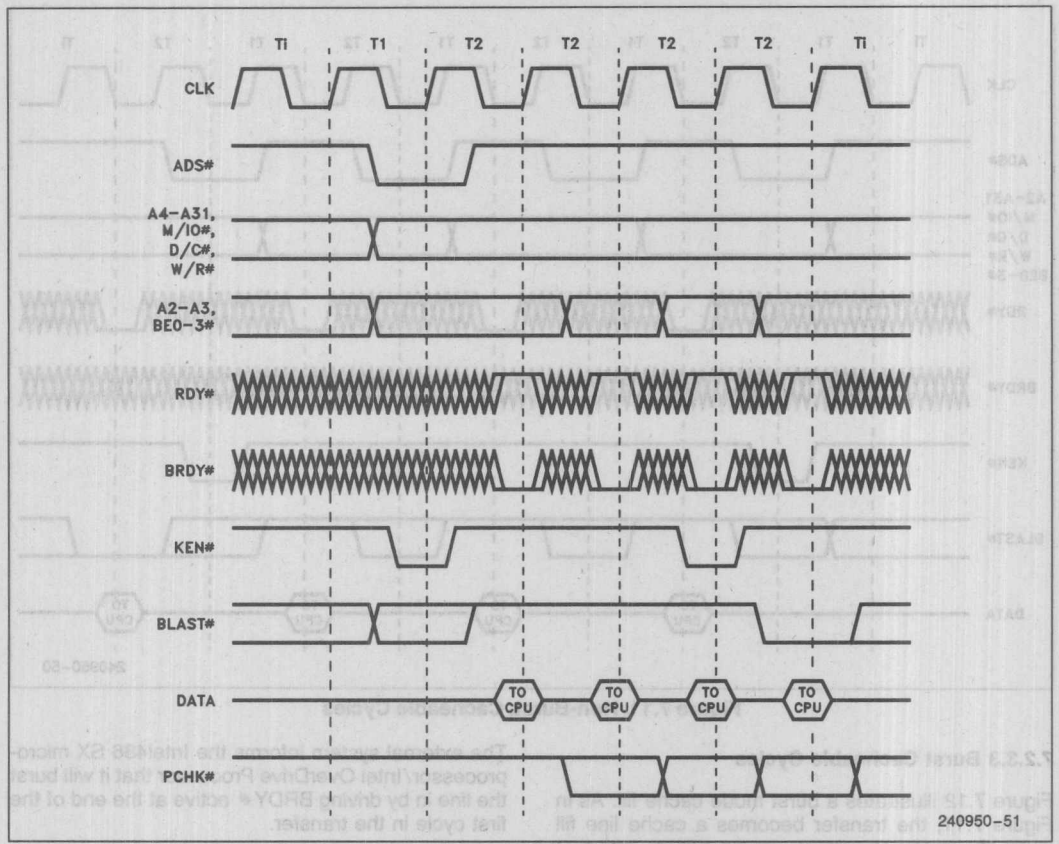


Figure 7.12. Burst Cacheable Cycle

7.2.3.4 Effect of Changing KEN# during a Cache Line Fill

KEN# can change multiple times as long as it arrives at its final value in the clock before RDY# or BRDY# is returned. This is illustrated in Figure 7.13. Note that the timing of BLAST# follows that of KEN# by one clock. The Intel486 SX microprocessor/Intel OverDrive Processor samples KEN# every clock and uses the value returned in the clock before ready to determine if a bus cycle would be a cache line fill. Similarly, it uses the value of KEN# in the last cycle, before early RDY# to load the line just retrieved from the memory into the cache. KEN# is sampled every clock, it must satisfy setup and hold time.

KEN# can also change multiple times before a burst cycle as long as it arrives at its final value one clock before ready is returned active.

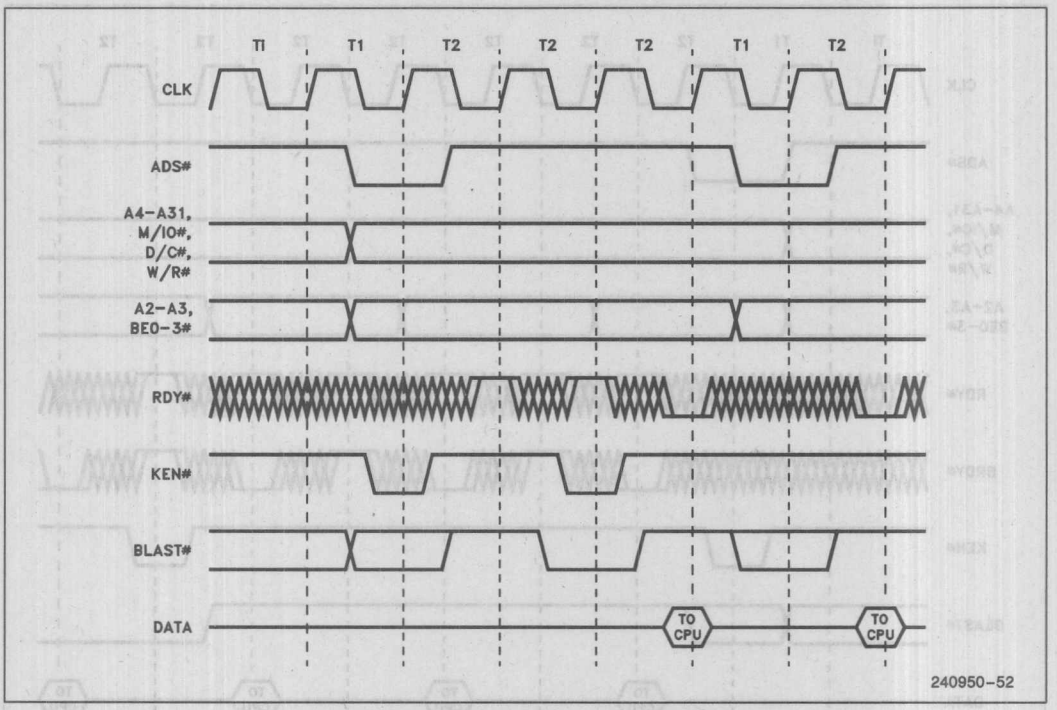


Figure 7.13. Effect of Changing KEN#

7.2.4 BURST MODE DETAILS

7.2.4.1 Adding Wait States to Burst Cycles

Burst cycles need not return data on every clock. The Intel486 SX microprocessor/Intel OverDrive Processor will only strobe data into the chip when either RDY# or BRDY# are active. Driving BRDY# and RDY# inactive adds a wait state to the transfer. A burst cycle where two clocks are required for every burst item is shown in Figure 7.14.

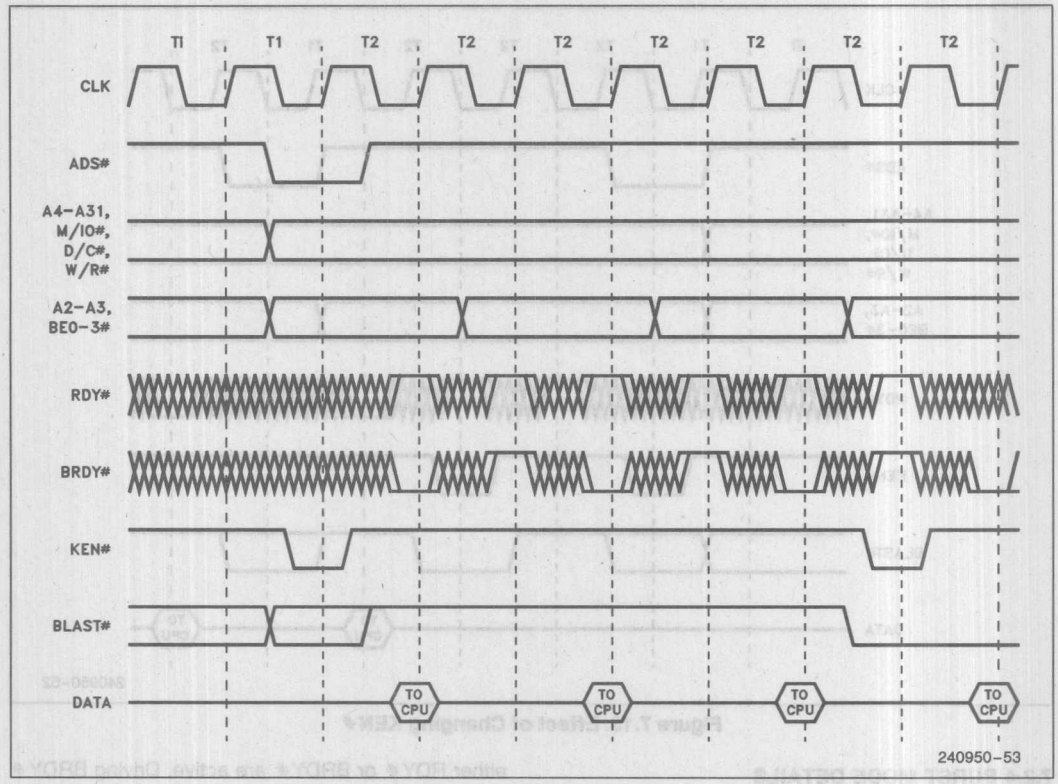


Figure 7.14. Slow Burst Cycle

7.2.4.2 Burst and Cache Line Fill Order

The burst order used by the Intel486 SX microprocessor/Intel OverDrive Processor is shown in Table 7.7. This burst order is followed by any burst cycle (cache or not), cache line fill (burst or not) or code prefetch.

The Intel486 SX microprocessor/Intel OverDrive Processor presents each request for data in an order determined by the first address in the transfer. For example, if the first address was 104 the next three addresses in the burst will be 100, 10C and 108.

Table 7.7. Burst Order

First Addr.	Second Addr.	Third Addr.	Fourth Addr.
0	4	8	C
4	0	C	8
8	C	0	4
C	8	4	0

An example of burst address sequencing is shown in Figure 7.15.

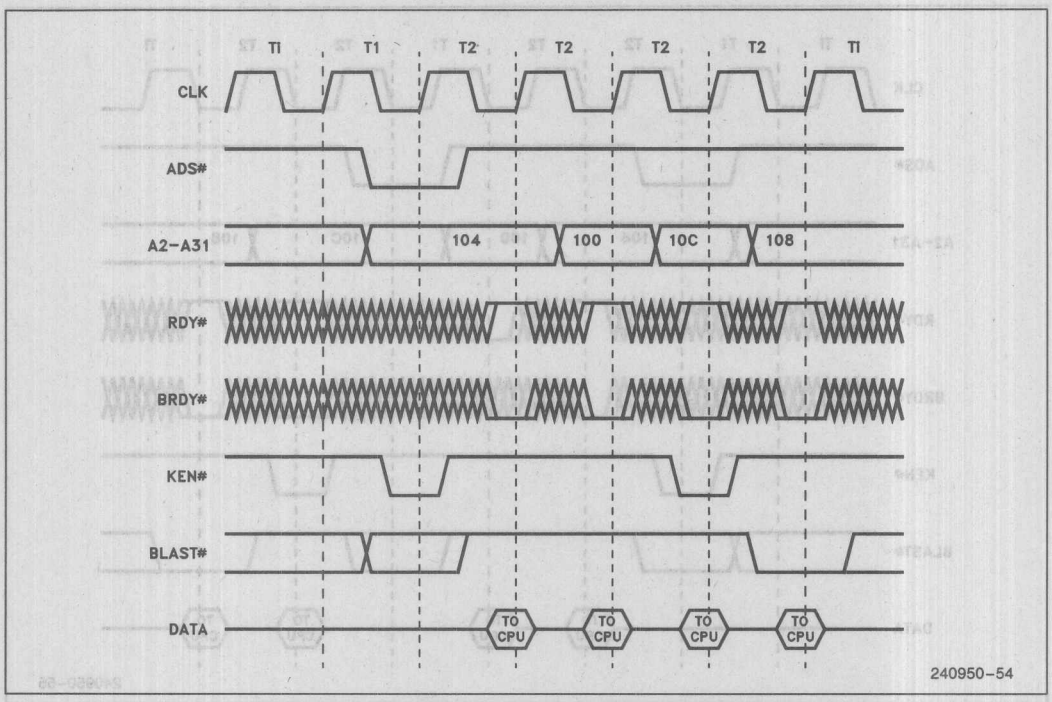


Figure 7.15. Burst Cycle Showing Order of Addresses

The sequences shown in Table 7.7 accommodate systems with 64-bit busses as well as systems with 32-bit data busses. The sequence applies to all bursts, regardless of whether the purpose of the burst is to fill a cache line, do a 64-bit read, or do a pre-fetch. If either BS8# or BS16# is returned active, the Intel486 SX microprocessor/Intel OverDrive Processor completes the transfer of the current 32-bit word before progressing to the next 32-bit word. For example, a BS16# burst to address 4 has the following order: 4-6-0-2-C-E-8-A.

7.2.4.3 Interrupted Burst Cycles

Some memory systems may not be able to respond with burst cycles in the order defined in Table 7.7. To support these systems the Intel486 SX microprocessor/Intel OverDrive Processor allows a burst cycle to be interrupted at any time. The Intel486 SX microprocessor/Intel OverDrive Processor will automatically generate another normal bus cycle after being interrupted to complete the data transfer. This is called an interrupted burst cycle. The external system can respond to an interrupted burst cycle with another burst cycle.

The external system can interrupt a burst cycle by returning RDY# instead of BRDY#. RDY# can be returned after any number of data cycles terminated with BRDY#.

An example of an interrupted burst cycle is shown in Figure 7.16. The Intel486 SX microprocessor/Intel OverDrive Processor immediately drives ADS# active to initiate a new bus cycle after RDY# is returned active. BLAST# driven inactive one clock after ADS# begins the second bus cycle indicating that the transfer is not complete.

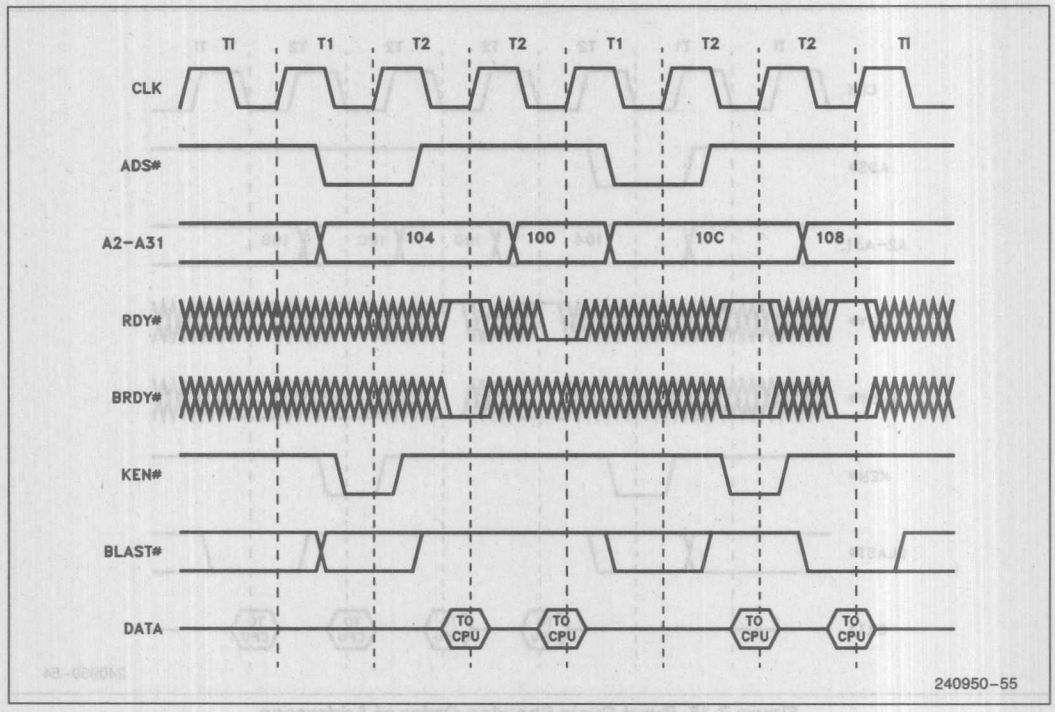


Figure 7.16. Interrupted Burst Cycle

KEN# need not be returned active in the first data cycle of the second part of the transfer in Figure 7.16. The cycle had been converted to a cache fill in the first part of the transfer and the Intel486 SX microprocessor/Intel OverDrive Processor expects the cache fill to be completed. Note that the first half and second half of the transfer in Figure 7.16 are each two cycle burst transfers.

The order in which the Intel486 SX microprocessor/ Intel OverDrive Processor requests operands during an interrupted burst transfer is determined by Table 7.7. Mixing RDY# and BRDY# does not change the order in which operand addresses are requested by the Intel486 SX microprocessor/Intel OverDrive Processor.

An example of the order in which the Intel486 SX microprocessor/Intel OverDrive Processor requests operands during a cycle in which the external system mixes RDY# and BRDY# is shown in Figure 7.17. The Intel486 SX microprocessor/Intel OverDrive Processor initially requests a transfer beginning at location 104. The transfer becomes a cache line fill when the external system returns KEN# active. The first cycle of the cache fill transfers the contents of location 104 and is terminated with RDY#. The Intel486 SX microprocessor/Intel OverDrive Processor drives out a new request (by asserting ADS#) to address 100. If the external system terminates the second cycle with BRDY#, the Intel486 SX microprocessor/Intel OverDrive Processor will next request/expect address 10C. The correct order is determined by the first cycle in the transfer, which may not be the first cycle in the burst if the system mixes RDY# with BRDY#.

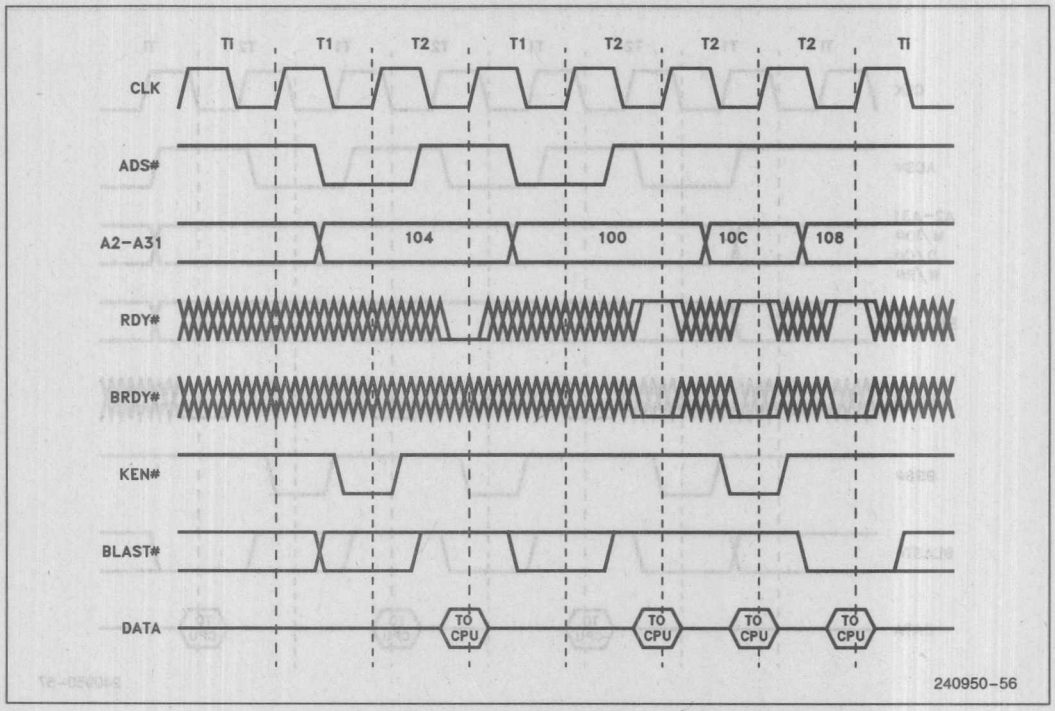


Figure 7.17. Interrupted Burst Cycle with Unobvious Order of Addresses

7.2.5 8- AND 16-BIT CYCLES

The Intel486 SX microprocessor/Intel OverDrive Processor supports both 16- and 8-bit external busses through the BS16# and BS8# inputs. BS16# and BS8# allow the external system to specify, on a cycle by cycle basis, whether the addressed component can supply 8, 16 or 32 bits. BS16# and BS8# can be used in burst cycles as well as non-burst cycles. If both BS16# and BS8# are returned active for any bus cycle, the Intel486 SX microprocessor/Intel OverDrive Processor will respond as if only BS8# were active.

The timing of BS16# and BS8# is the same as that of KEN#. BS16# and BS8# must be driven active before the first RDY# or BRDY# is driven active.

Driving the BS16# and BS8# active can force the Intel486 SX microprocessor/Intel OverDrive Processor to run additional cycles to complete what would have been only a single 32-bit cycle. BS8# and BS16# may change the state of BLAST# when they force subsequent cycles from the transfer.

Figure 7.18 shows an example in which BS8# forces the Intel486 SX microprocessor/Intel OverDrive Processor to run two extra cycles to complete a transfer. The Intel486 SX microprocessor/Intel OverDrive Processor issues a request for 24 bits of information. The external system drives BS8# active indicating that only eight bits of data can be supplied per cycle. The Intel486 SX microprocessor/Intel OverDrive Processor issues two extra cycles to complete the transfer.

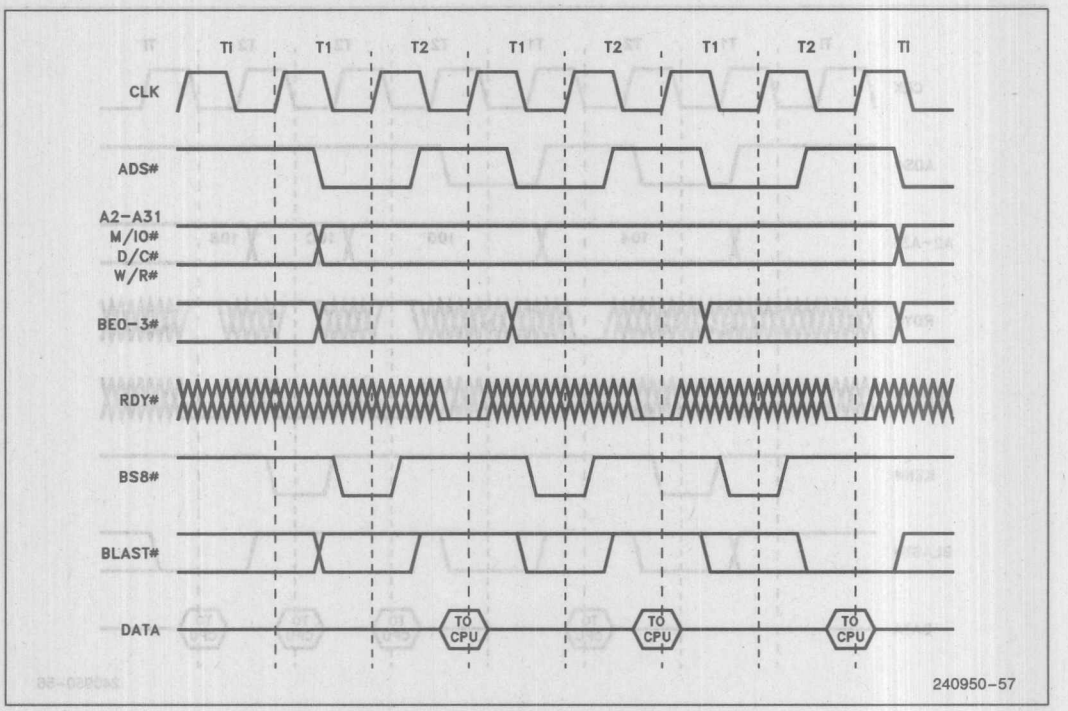


Figure 7.18. 8-Bit Bus Size Cycle

Extra cycles forced by the BS16# and BS8# should be viewed as independent bus cycles. BS16# and BS8# should be driven active for each additional cycle unless the addressed device has the ability to change the number of bytes it can return between cycles. The Intel486 SX microprocessor/Intel OverDrive Processor will drive BLAST# inactive until the last cycle before the transfer is complete.

Refer to Section 7.1.3 for the sequencing of addresses while BS8# or BS16# are active.

BS8# and BS16# operate during burst cycles in exactly the same manner as non-burst cycles. For example, a single non-cacheable read could be transferred by the Intel486 SX microprocessor/Intel OverDrive Processor as four 8-bit burst data cycles. Similarly, a single 32-bit write could be written as four 8-bit burst data cycles. An example of a burst write is shown in Figure 7.19. Burst writes can only occur if BS8# or BS16# is asserted.

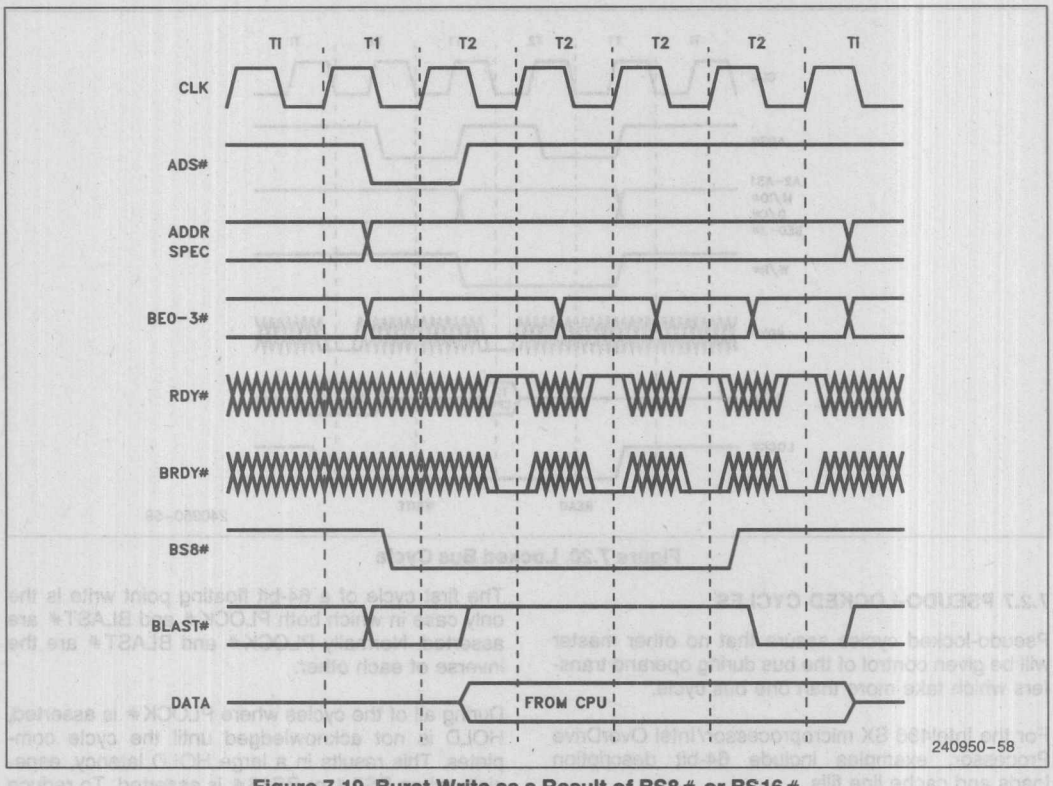


Figure 7.19. Burst Write as a Result of BS8# or BS16#

7.2.6 LOCKED CYCLES

Locked cycles are generated in software for any instruction that performs a read-modify-write operation. During a read-modify-write operation the Intel486 SX microprocessor/Intel OverDrive Processor can read and modify a variable in external memory and be assured that the variable is not accessed between the read and write.

Locked cycles are automatically generated during certain bus transfers. The xchg (exchange) instruction generates a locked cycle when one of its operands is memory based. Locked cycles are generated when a segment or page table entry is updated and during interrupt acknowledge cycles. Locked cycles are also generated when the LOCK instruction prefix is used with selected instructions.

Locked cycles are implemented in hardware with the LOCK# pin. When LOCK# is active, the Intel486 SX microprocessor/Intel OverDrive Processor is performing a read-modify-write operation and the external bus should not be relinquished until the cycle is complete. Multiple reads or writes can be locked. A locked cycle is shown in Figure 7.20. LOCK# goes active with the address and bus definition pins at the beginning of the first read cycle and remains active until RDY# is returned for the last write cycle. For unaligned 32 bits read-modify-write operation, the LOCK# remains active for the entire duration of the multiple cycle. It will go inactive when RDY# is returned for the last write cycle.

When LOCK# is active, the Intel486 SX microprocessor/Intel OverDrive Processor will recognize address hold and backoff but will not recognize bus hold. It is left to the external system to properly arbitrate a central bus when the Intel486 SX microprocessor/Intel OverDrive Processor generates LOCK#.

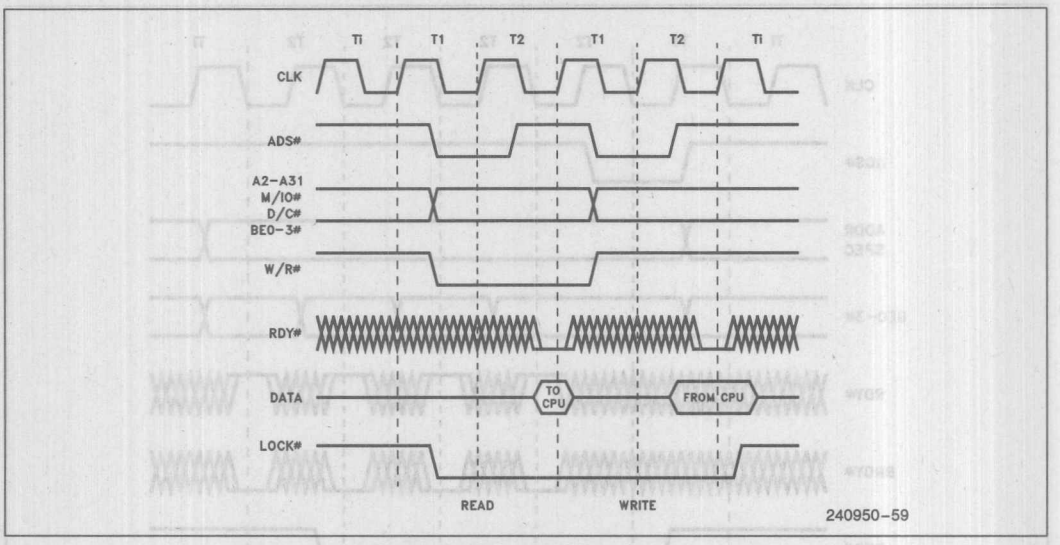


Figure 7.20. Locked Bus Cycle

7.2.7 PSEUDO-LOCKED CYCLES

Pseudo-locked cycles assure that no other master will be given control of the bus during operand transfers which take more than one bus cycle.

For the Intel486 SX microprocessor/Intel OverDrive Processor, examples include 64-bit description loads and cache line fills.

For the Intel OverDrive Processor only, 64-bit floating point read and write cycles are also examples of operand transfers which take more than one bys cycle.

Pseudo-locked transfers are indicated by the PLOCK# pin. The memory operands must be aligned for correct operation of a pseudo-locked cycle.

PLOCK# need not be examined during burst reads. A 64-bit aligned operand can be retrieved in one burst (note: this is only valid in systems that do not interrupt bursts).

The system must examine PLOCK# during 64-bit writes since the Intel486 SX microprocessor/Intel OverDrive Processor cannot burst write more than 32 bits. However, burst can be used within each 32-bit write cycle if BS8# or BS16# is asserted. BLAST will be deasserted in response to BS8# or BS16#. A 64-bit write will be driven out as two non-burst bus cycles. BLAST# is asserted during both writes since a burst is not possible. PLOCK# is asserted during the first write to indicate that another write follows. This behavior is shown in Figure 7.21.

The first cycle of a 64-bit floating point write is the only case in which both PLOCK# and BLAST# are asserted. Normally PLOCK# and BLAST# are the inverse of each other.

During all of the cycles where PLOCK# is asserted, HOLD is not acknowledged until the cycle completes. This results in a large HOLD latency, especially when BS8# or BS16# is asserted. To reduce the HOLD latency during these cycles, windows are available between transfers to allow HOLD to be acknowledged during non-cacheable code prefetches. PLOCK# will be asserted since BLAST# is negated, but it is ignored and HOLD is recognized during the prefetch.

PLOCK# can change several times during a cycle settling to its final value in the clock ready is returned.

7.2.8 INVALIDATE CYCLES

Invalidate cycles are needed to keep the Intel486 SX microprocessor/Intel OverDrive Processor internal cache contents consistent with external memory. The Intel486 SX microprocessor/Intel OverDrive Processor contains a mechanism for listening to writes by other devices to external memory. When the Intel486 SX microprocessor/Intel OverDrive Processor finds a write to a Section of external memory contained in its internal cache, the Intel486 SX microprocessor/Intel OverDrive Processor's internal copy is invalidated.

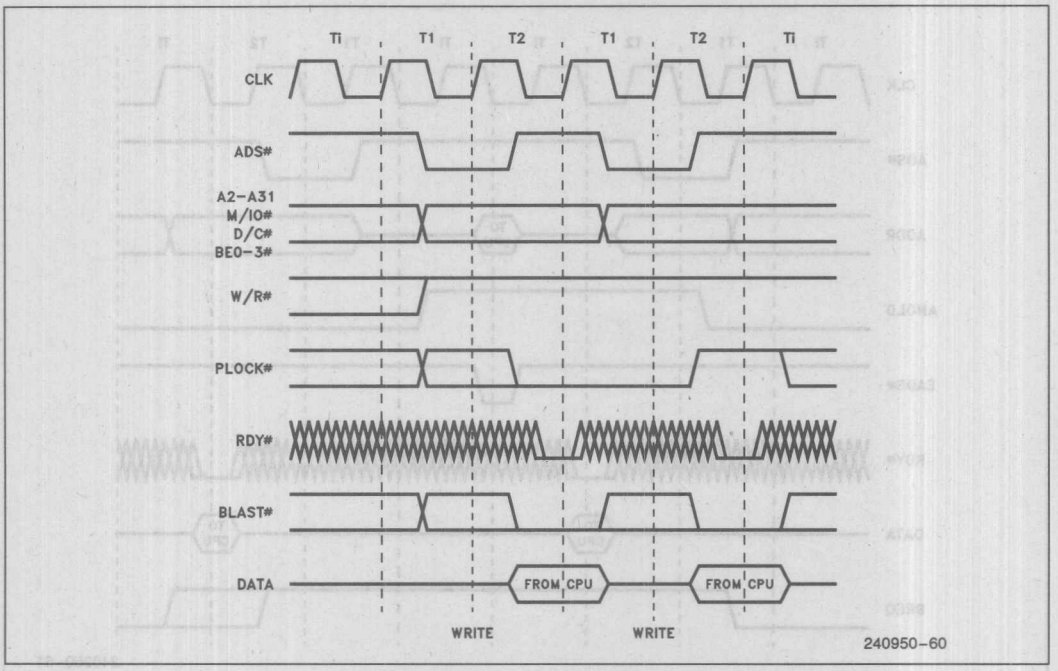


Figure 7.21. Pseudo Lock Timing

Invalidations use two pins, address hold request (AHOLD) and valid external address (EADS#). There are two steps in an invalidation cycle. First, the external system asserts the AHOLD input forcing the Intel486 SX microprocessor/Intel OverDrive Processor to immediately relinquish its address bus. Next, the external system asserts EADS# indicating that a valid address is on the Intel486 SX microprocessor/Intel OverDrive Processor address bus. Figure 7-22 shows the fastest possible invalidation cycle. The Intel486 SX microprocessor/Intel OverDrive Processor recognizes AHOLD on one CLK edge and floats the address bus in response. To allow the address bus to float and avoid contention, EADS# and the invalidation address should not be driven until the following CLK edge. The Intel486 SX microprocessor/Intel OverDrive Processor reads the address over its address lines. If the Intel486 SX microprocessor/Intel OverDrive Processor finds this address in its internal cache, the cache entry is invalidated. Note that the Intel486 SX microprocessor/Intel OverDrive Processor address bus is input/output unlike the Intel386 microprocessor's bus, which is output only.

The Intel486 SX microprocessor/Intel OverDrive Processor immediately relinquishes its address bus in the next clock upon assertion of AHOLD. For example, the bus could be 3 wait states into a read cycle. If AHOLD is activated, the Intel486 SX microprocessor/Intel OverDrive Processor will immediately float its address bus before ready is returned terminating the bus cycle.

When AHOLD is asserted only the address bus is floated, the data bus can remain active. Data can be returned for a previously specified bus cycle during address hold (see Figures 7.22, 7.23).

EADS# is normally asserted when an external master drives an address onto the bus. AHOLD need not be driven for EADS# to generate an internal invalidate. If EADS# alone is asserted while the Intel486 SX microprocessor/Intel OverDrive Processor is driving the address bus, it is possible that the invalidation address will come from the Intel486 SX microprocessor/Intel OverDrive Processor itself.

Note that it is also possible to run an invalidation cycle by asserting EADS# when HOLD or BUFF# is asserted.

Running an invalidate cycle prevents the Intel486 SX microprocessor/Intel OverDrive Processor cache from satisfying other internal requests, so invalidations should be run only when necessary. The fastest possible invalidate cycle is shown in Figure 7.22, while a more realistic invalidation cycle is shown in 7.23. Both of the examples take one clock of cache access from the Intel486 SX microprocessor/Intel OverDrive Processor.

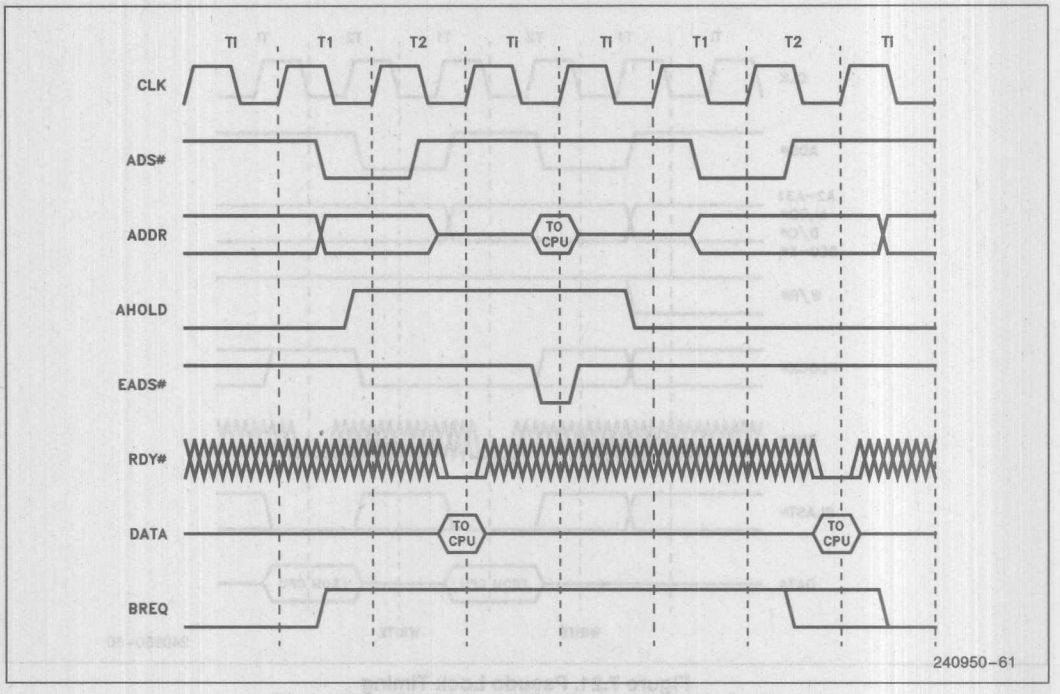


Figure 7.22. Fast Internal Cache Invalidation Cycle

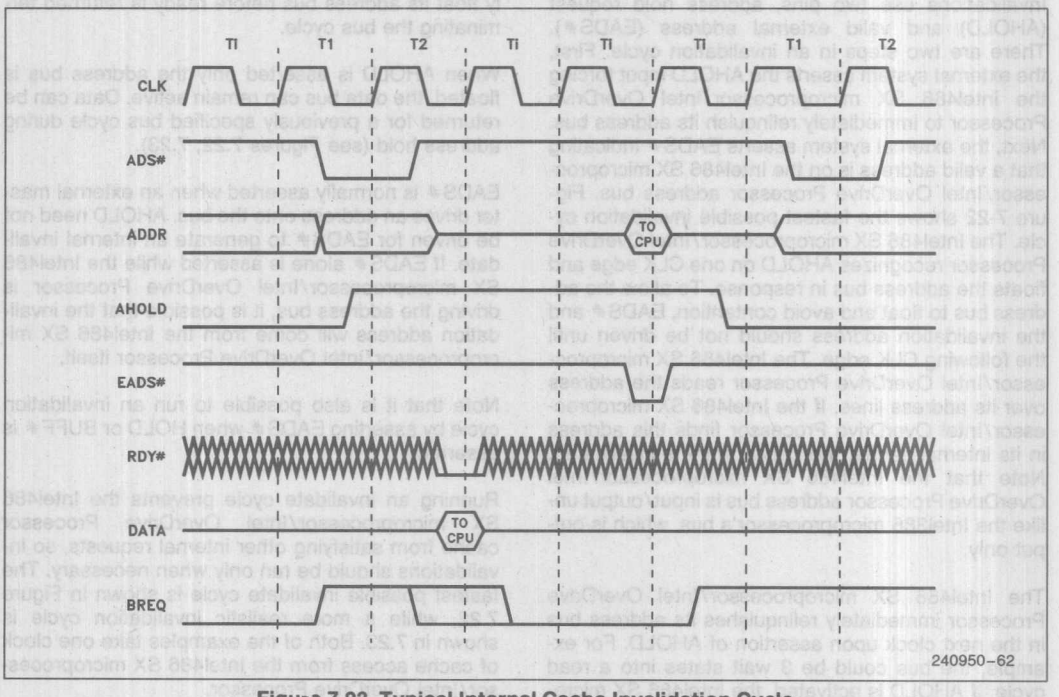


Figure 7.23. Typical Internal Cache Invalidation Cycle

7.2.8.1 Rate of Invalidate Cycles

The Intel486 SX microprocessor/Intel OverDrive Processor can accept one invalidate per clock except in the last clock of a line fill. One invalidate per clock is possible as long as EADS# is negated in ONE or BOTH of the following cases:

1. In the clock RDY# or BRDY# is returned for the last time.
2. In the clock following RDY# or BRDY# being returned for the last time.

This definition allows two system designs. Simple designs can restrict invalidates to one every other clock. The simple design need not track bus activity. Alternatively, systems can request one invalidate per clock provided that the bus is monitored.

7.2.8.2 Running Invalidate Cycles Concurrently with Line Fills

Precautions are necessary to avoid caching stale data in the Intel486 SX microprocessor/Intel OverDrive Processor cache in a system with a second level cache. An example of a system with a second level cache is shown in Figure 7.24. An external device can be writing to main memory over the system bus while the Intel486 SX microprocessor/Intel OverDrive Processor is retrieving data from the second level cache. The Intel486 SX microprocessor/Intel OverDrive Processor will need to invalidate a line in its internal cache if the external device is writing to a main memory address also contained in the Intel486 SX microprocessor/Intel OverDrive Processor cache.

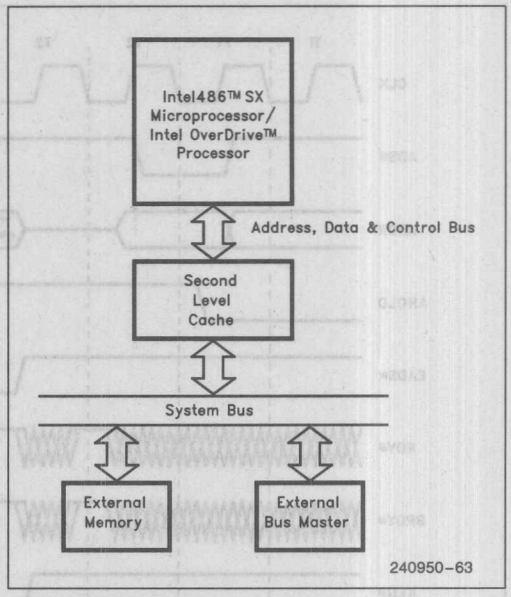
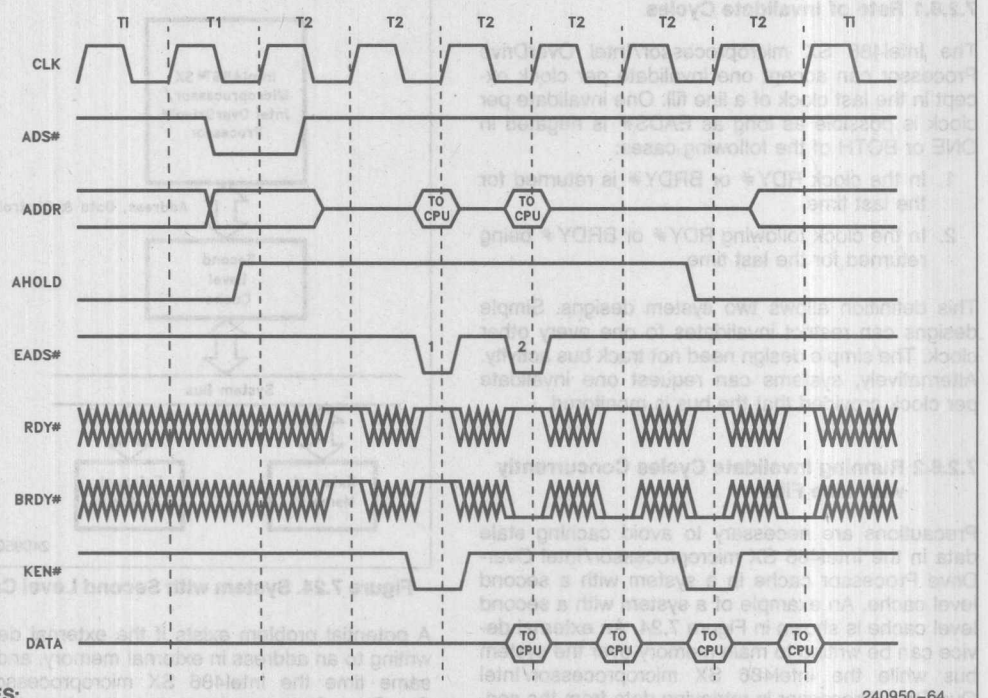


Figure 7.24. System with Second Level Cache

A potential problem exists if the external device is writing to an address in external memory, and at the same time the Intel486 SX microprocessor/Intel OverDrive Processor is reading data from the same address in the second level cache. The system must force an invalidation cycle to invalidate the data that the Intel486 SX microprocessor/Intel OverDrive Processor has requested during the line fill.

If the system asserts EADS# before the first data in the line fill is returned to the Intel486 SX microprocessor/Intel OverDrive Processor, the system must return data consistent with the new data in the external memory upon resumption of the line fill after the invalidation cycle. This is illustrated by the asserted EADS# signal labeled 1 in Figure 7.25.



NOTES:
1. Data returned must be consistent if its address equals the invalidation address in this clock
2. Data returned will not be cached if its address equals the invalidation address in this clock

240950-64

Figure 7.25. Cache Invalidation Cycle Concurrent with Line Fill

If the system asserts EADS# at the same time or after the first data in the line fill is returned (in the same clock that the first RDY# or BRDY# is returned or any subsequent clock in the line fill) the data will be read into the Intel486 SX microprocessor/Intel OverDrive Processor input buffers but it will not be stored in the on-chip cache. This is illustrated by asserted EADS# signal labeled 2 in Figure 7.25. The stale data will be used to satisfy the request that initiated the cache fill cycle.

7.2.9 BUS HOLD

The Intel486 SX microprocessor/Intel OverDrive Processor provides a bus hold, hold acknowledge protocol using the bus hold request (HOLD) and bus hold acknowledge (HLDA) pins. Asserting the HOLD input indicates that another bus master desires control of the Intel486 SX microprocessor/Intel OverDrive Processor bus. The Intel486 SX microprocessor/Intel OverDrive Processor will respond by floating its bus and driving HLDA active when the current bus cycle, or sequence of locked cycles is complete. An example of a HOLD/HLDA transaction is shown in Figure 7.26a. Unlike the Intel386 microprocessor, the Intel486 SX microprocessor/Intel OverDrive Processor can respond to HOLD by floating its bus and asserting HLDA while RESET is asserted.

Note that HOLD will be recognized during un-aligned writes (less than or equal to 32-bits) with BLAST# being active for each write. For greater than 32-bit or un-aligned write, HOLD# recognition is prevented by PLOCK# getting asserted. However, HOLD is recognized during non-cachable, non-burstable code prefetches even though PLOCK# is active.

For cacheable and nonbursted or bursted cycles, HOLD is acknowledged during backoff only if HOLD and BOFF# are asserted during an active bus cycle (after ADS# asserted) and before the first RDY# or BRDY# has been returned (see Figure 7.26b). The order in which HOLD and BOFF# go active is unim-

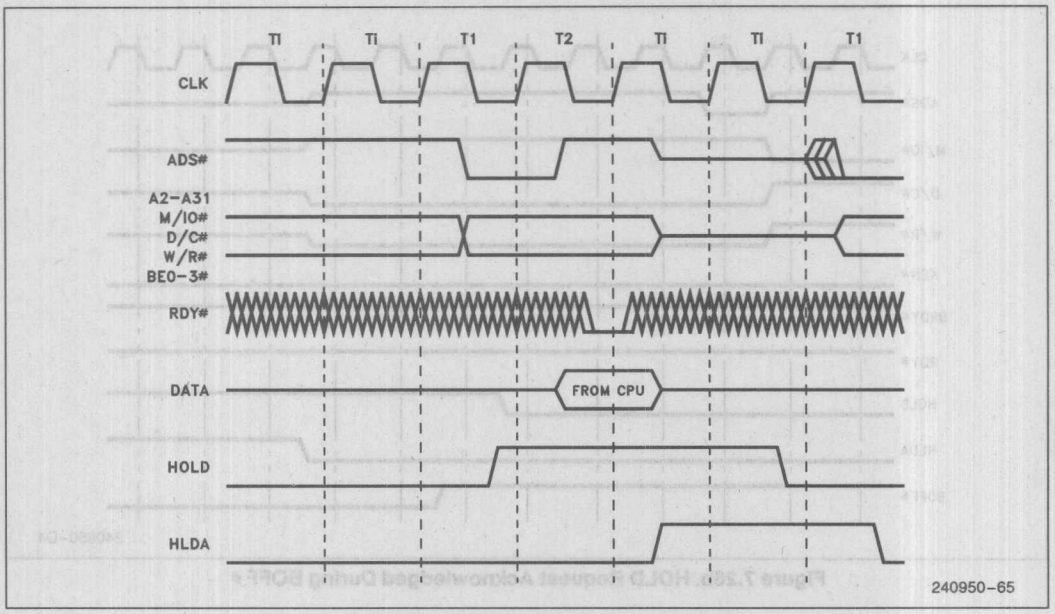


Figure 7.26a. HOLD/HLDA Cycles

portant (so long as both are active prior to the first RDY#/BRDY# returned by the system). Figure 7.26b shows the case where HOLD is asserted first; HOLD could be asserted simultaneously or after BOFF# and still be acknowledged.

The pins floated during bus hold are: BE0#–BE3#, PCD, PWT, W/R#, D/C#, M/IO#, LOCK#, PLOCK#, ADS#, BLAST#, D0–D31, A2–A31, DP0–DP3.

7.2.10 INTERRUPT ACKNOWLEDGE

The Intel486 SX microprocessor/Intel OverDrive Processor generates interrupt acknowledge cycles in response to maskable interrupt requests generated on the interrupt request input (INTR) pin. Interrupt acknowledge cycles have a unique cycle type generated on the cycle type pins.

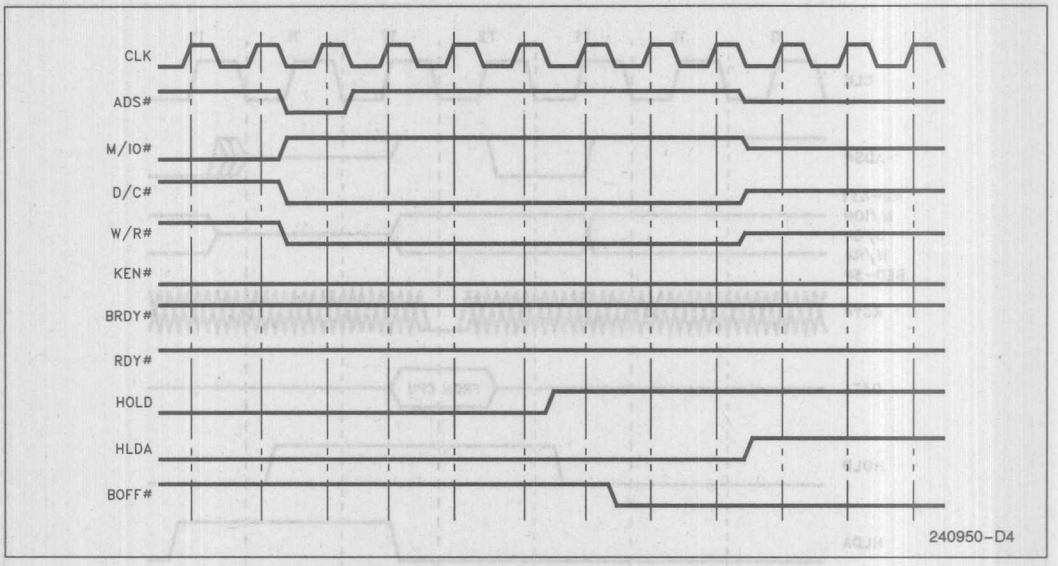


Figure 7.26b. HOLD Request Acknowledged During BOFF#

An example of an interrupt acknowledge transaction is shown in Figure 7.27. Interrupt acknowledge cycles are generated in locked pairs. Data returned during the first cycle is ignored. The interrupt vector is returned during the second cycle on the lower 8 bits of the data bus. The Intel486 SX microprocessor/Intel OverDrive Processor has 256 possible interrupt vectors.

The state of A2 distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A31–A3 low, A2 high, BE3#–BE1# high, and BE0# low). The address driven during the second interrupt acknowledge cycle is 0 (A31–A2 low, BE3#–BE1# high, BE0# low).

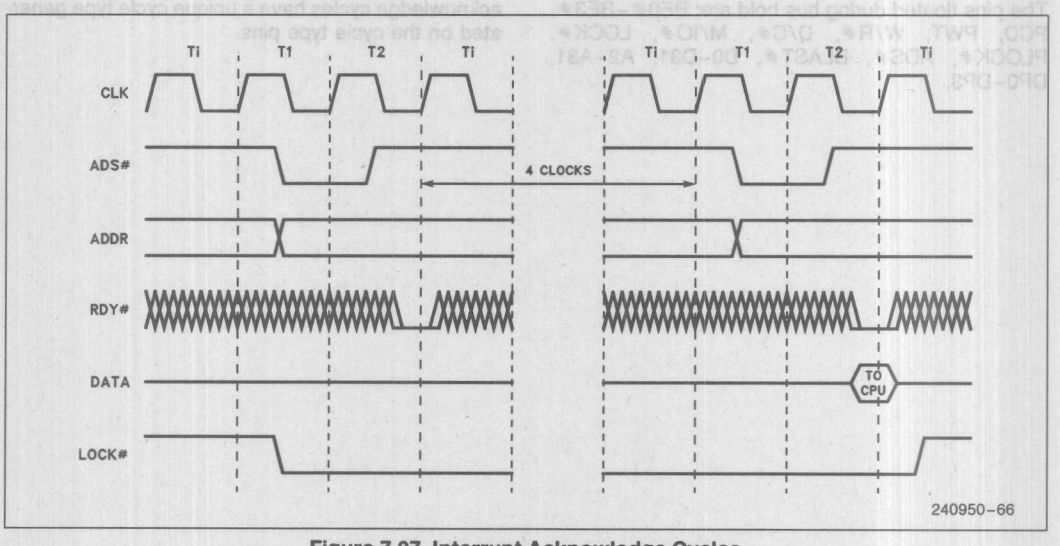


Figure 7.27. Interrupt Acknowledge Cycles

Each of the interrupt acknowledge cycles are terminated when the external system returns RDY# or BRDY#. Wait states can be added by withholding RDY# or BRDY#. The Intel486 SX microprocessor/Intel OverDrive Processor automatically generates four idle clocks between the first and second cycles to allow for 8259A recovery time.

7.2.11 SPECIAL BUS CYCLES

The Intel486 SX microprocessor/Intel OverDrive Processor provides four special bus cycles to indicate that certain instructions have been executed, or certain conditions have occurred internally. The special bus cycles in Table 7.8 are defined when the bus cycle definition pins are in the following state: M/IO# = 0, D/C# = 0 and W/R# = 1. During these cycles the address bus is driven low while the data bus is undefined.

Two of the special cycles indicate halt or shutdown. Another special cycle is generated when the Intel486 SX microprocessor/Intel OverDrive Processor executes an INVD (invalidate data cache) instruction and could be used to flush an external cache. The Write Back cycle is generated when the Intel486 SX microprocessor/Intel OverDrive Processor executes the WBINVD (write-back invalidate data cache) instruction and could be used to synchronize an external write-back cache.

The external hardware must acknowledge these special bus cycles by returning RDY# or BRDY#.

Table 7.8. Special Bus Cycle Encoding

BE3#	BE2#	BE1#	BE0#	Special Bus Cycle
1	1	1	0	Shutdown
1	1	0	1	Flush
1	0	1	1	Halt
0	1	1	1	Write Back

7.2.11.1 Halt Indication Cycle

The Intel486 SX microprocessor/Intel OverDrive Processor halts as a result of executing a HALT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the bus definition signals in special bus cycle state and a byte address of 2. BE0# and BE2# are the only signals distinguishing halt indication from shutdown indication, which drives an address of 0. During the halt cycle undefined data is driven on D0–D31. The halt indication cycle must be acknowledged by READY# asserted.

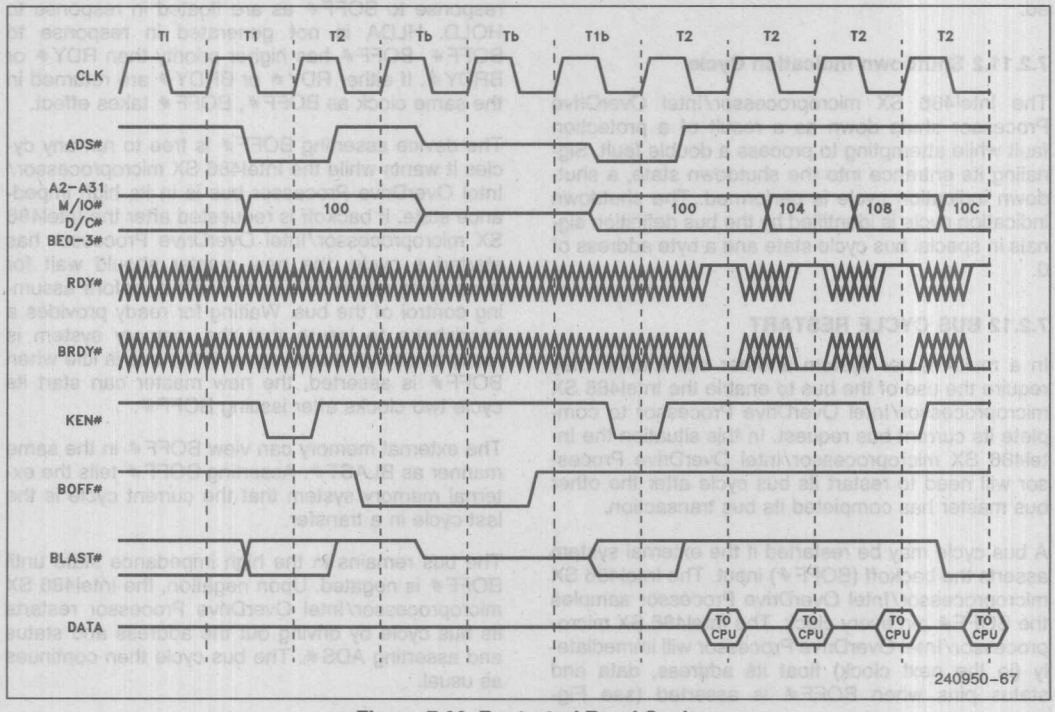


Figure 7.28. Restarted Read Cycle

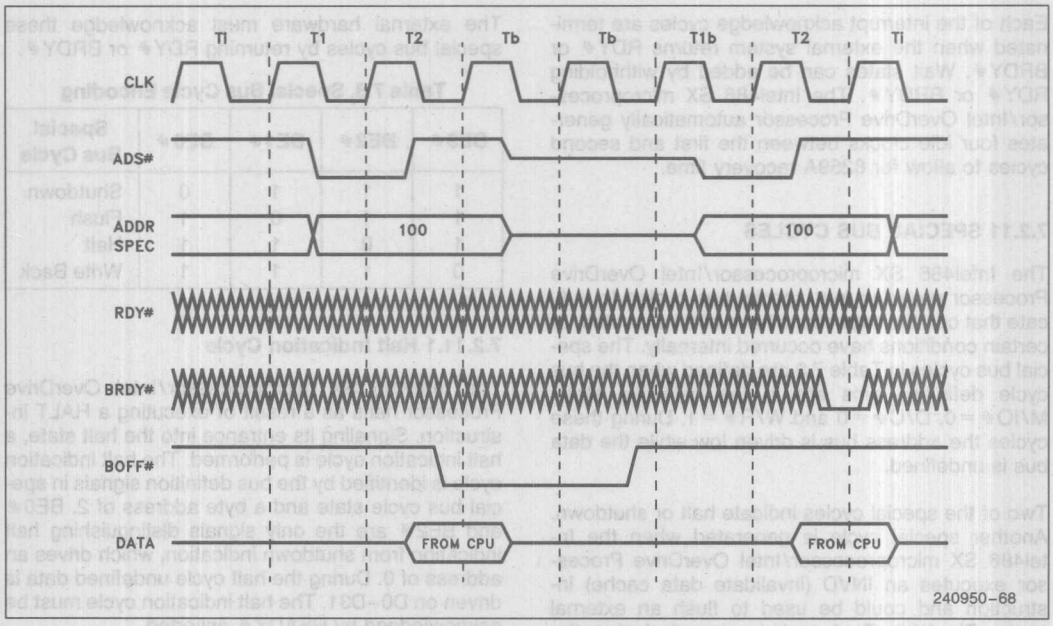


Figure 7.29. Restarted Write Cycle

A halted Intel486 SX microprocessor/Intel OverDrive Processor resumes execution when INTR (if interrupts are enabled) or NMI or RESET is asserted.

7.2.11.2 Shutdown Indication Cycle

The Intel486 SX microprocessor/Intel OverDrive Processor shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the bus definition signals in special bus cycle state and a byte address of 0.

7.2.12 BUS CYCLE RESTART

In a multi-master system another bus master may require the use of the bus to enable the Intel486 SX microprocessor/Intel OverDrive Processor to complete its current bus request. In this situation the Intel486 SX microprocessor/Intel OverDrive Processor will need to restart its bus cycle after the other bus master has completed its bus transaction.

A bus cycle may be restarted if the external system asserts the backoff (BOFF#) input. The Intel486 SX microprocessor/Intel OverDrive Processor samples the BOFF# pin every clock. The Intel486 SX microprocessor/Intel OverDrive Processor will immediately (in the next clock) float its address, data and status pins when BOFF# is asserted (see Figure 7.28). Any bus cycle in progress when BOFF# is asserted is aborted and any data returned to the processor is ignored. The same pins are floated in response to BOFF# as are floated in response to HOLD. HLDA is not generated in response to BOFF#. BOFF# has higher priority than RDY# or BRDY#. If either RDY# or BRDY# are returned in the same clock as BOFF#, BOFF# takes effect.

The device asserting BOFF# is free to run any cycles it wants while the Intel486 SX microprocessor/Intel OverDrive Processor bus is in its high impedance state. If backoff is requested after the Intel486 SX microprocessor/Intel OverDrive Processor has started a cycle, the new master should wait for memory to return RDY# or BRDY# before assuming control of the bus. Waiting for ready provides a handshake to insure that the memory system is ready to accept a new cycle. If the bus is idle when BOFF# is asserted, the new master can start its cycle two clocks after issuing BOFF#.

The external memory can view BOFF# in the same manner as BLAST#. Asserting BOFF# tells the external memory system that the current cycle is the last cycle in a transfer.

The bus remains in the high impedance state until BOFF# is negated. Upon negation, the Intel486 SX microprocessor/Intel OverDrive Processor restarts its bus cycle by driving out the address and status and asserting ADS#. The bus cycle then continues as usual.

Asserting BOFF# during a burst, BS8# or BS16# cycle will force the Intel486 SX microprocessor/Intel OverDrive Processor to ignore data returned for that cycle only. Data from previous cycles will still be valid. For example, if BOFF# is asserted on the third BRDY# of a burst, the Intel486 SX microprocessor/Intel OverDrive Processor assumes the data returned with the first and second BRDY#'s is correct and restarts the burst beginning with the third item. The same rule applies to transfers broken into multiple cycle by BS8# or BS16#.

Asserting BOFF# in the same clock as ADS# will cause the Intel486 SX microprocessor/Intel OverDrive Processor to float its bus in the next clock and leave ADS# floating low. Since ADS# is floating low, a peripheral may think that a new bus cycle has begun even-though the cycle was aborted. There are two possible solutions to this problem. The first is to have all devices recognize this condition and ignore ADS# until ready comes back. The second approach is to use a "two clock" backoff: in the first clock AHOLD is asserted, and in the second clock BOFF# is asserted. This guarantees that ADS# will not be floating low. This is only necessary in systems where BOFF# may be asserted in the same clock as ADS#.

7.2.13 BUS STATES

A bus state diagram is shown in Figure 7.30. A description of the signals used in the diagram is given in Table 7.9.

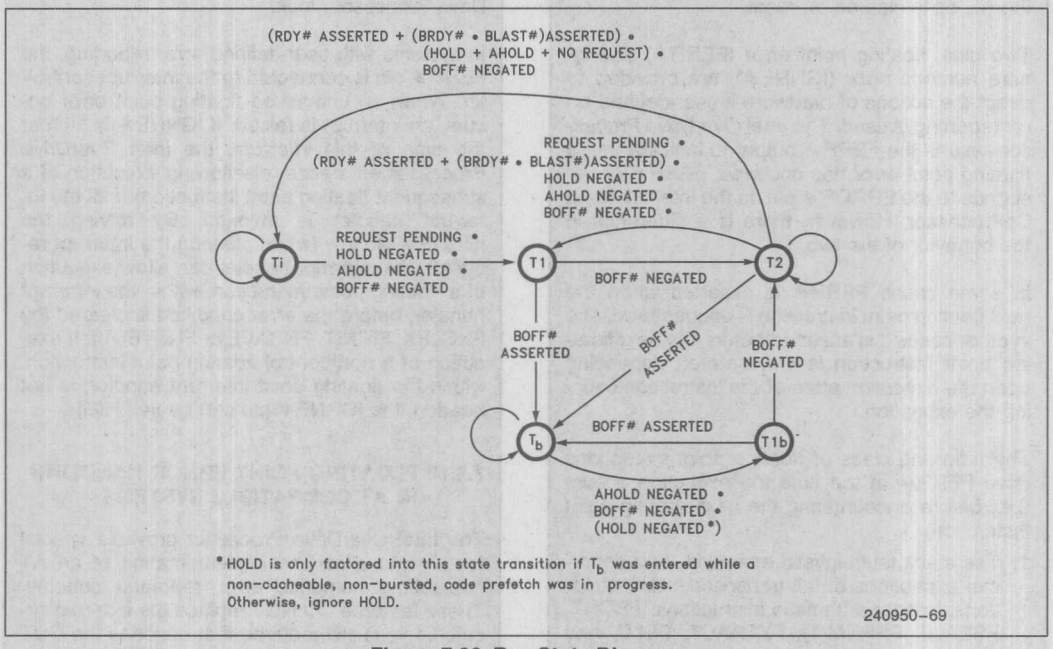


Figure 7.30. Bus State Diagram

Table 7.9. Bus State Description

State	Means
Ti	Bus is idle. Address and status signals may be driven to undefined values, or the bus may be floated to a high impedance state.
T1	First clock cycle of a bus cycle. Valid address and status are driven and ADS# is asserted.
T2	Second and subsequent clock cycles of a bus cycle. Data is driven if the cycle is a write, or data is expected if the cycle is a read. RDY# and BRDY# are sampled.
T1b	First clock cycle of a restarted bus cycle. Valid address and status are driven and ADS# is asserted.
Tb	Second and subsequent clock cycles of an aborted bus cycle.

7.2.14 FLOATING POINT ERROR HANDLING

The Intel OverDrive Processor provides two options for reporting floating point errors. The simplest method is to raise interrupt 16 whenever an unmasked floating point error occurs. This option may be enabled by setting the NE bit in control register 0 (CR0).

The Intel OverDrive Processor also provides the option of allowing external hardware to determine how floating point errors are reported. This option is necessary for compatibility with the error reporting scheme used in DOS based systems. The NE bit must be cleared in CR0 to enable user-defined error reporting. User-defined error reporting is the default condition because the NE bit is cleared on reset.

Two pins, floating point error (FERR#) and ignore numeric error (IGNNE#), are provided to direct the actions of hardware if user-defined error reporting is used. The Intel OverDrive Processor asserts the FERR# output to indicate that a floating point error has occurred. FERR# corresponds to the ERROR# pin on the Intel387 Math CoProcessor. However, there is a difference in the behavior of the two.

In some cases FERR# is asserted when the next floating point instruction is encountered, and in other cases it is asserted before the next floating point instruction is encountered depending upon the execution state of the instruction causing the exception.

The following class of floating point exceptions drive FERR# at the time the exception occurs (i.e., before encountering the next floating point instruction).

1. The stack fault, invalid operation, and denormal exceptions on all transcendental instructions, integer arithmetic instructions, FSQRT, FSEALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exceptions on store instructions (including integer store instructions).

The following class of floating point exceptions drive FERR# only after encountering the next floating point instruction.

1. Exceptions other than on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exception on all basic arithmetic, load, compare, and control instructions (i.e., all other instructions).

For both sets of exceptions above, the 387 Math CoProcessor asserts ERROR# when the error occurs and does not wait for the next floating point instruction to be encountered.

IGNNE# is an input to the Intel OverDrive Processor When the NE bit in CR0 is cleared, and IGNNE# is asserted, the Intel OverDrive Processor will ignore a user floating point error and continue executing floating point instructions. When IGNNE# is negated, the IGNNE# is an input to the Intel OverDrive Processor will freeze on floating point instructions which get errors (except for the control instructions FNCLEX, FNINIT, FNSAVE, FNSTENV, FNSTCW, FNSTSW, FNSTSW AX, FNENI, FNDISI and FNSETPM). IGNNE# may be asynchronous to the Intel OverDrive Processor clock.

In systems with user-defined error reporting, the FERR# pin is connected to the interrupt controller. When an unmasked floating point error occurs, an interrupt is raised. If IGNNE# is high at the time of this interrupt, the Intel OverDrive Processor will freeze (disallowing execution of a subsequent floating point instruction) until the interrupt handler is invoked. By driving the IGNNE# pin low (when clearing the interrupt request), the interrupt handler can allow execution of a floating point instruction, within the interrupt handler, before the error condition is cleared (by FNCLEX, FNINIT, FNSAVE or FNSTENV). If execution of a non-control floating point instruction, within the floating point interrupt handler, is not needed, the IGNNE# pin can be tied HIGH.

7.2.15 FLOATING POINT ERROR HANDLING IN AT COMPATIBLE SYSTEMS

The Intel OverDrive Processor provides special features to allow the implementation of an AT compatible numerics error reporting scheme. These features DO NOT replace the external circuit. Logic is still required that decodes the OUT F0 instruction and latches the FERR# signal. What follows is a description of the use of these Intel OverDrive Processor features.

The features provided by the Intel OverDrive Processor are the NE bit in the Machine Status Register, the IGNNE# pin, and the FERR# pin.

The NE bit determines the action taken by the Intel OverDrive Processor when a numerics error is detected. When set this bit signals that non-DOS compatible error handling will be implemented. In this mode the Intel OverDrive Processor takes a software exception (16) if a numerics error is detected.

If the NE bit is reset the Intel OverDrive Processor uses the IGNNE# pin to allow an external circuit to control the time at which non-control numerics instructions are allowed to execute. Note that floating point control instructions such as FNINIT and FNSAVE can be executed during a floating point error condition regardless of the state of IGNNE#.

To process a floating point error in the DOS environment the following sequence must take place:

1. The error is detected by the Intel OverDrive Processor which activates the FERR# pin.
2. FERR# is latched so that it can be cleared by the OUT F0 instruction.
3. The latched FERR# signal activates an interrupt at the interrupt controller. This interrupt is usually handled on IRQ13.
4. The Interrupt Service Routine (ISR) handles the error and then clears the interrupt by executing an OUT instruction to port F0. The address F0 is decoded externally to clear the FERR# latch. The IGNNE# signal is also activated by the decoder output.
5. Usually the ISR then executes an FNINIT instruction or other control instruction before restarting the program. FNINIT clears the FERR# output.

Figure 7.31 illustrates the circuit required to perform this function. Note that this circuit has not been tested. It is included as an example of the required error handling logic.

Note that the IGNNE# input allows non-control instructions to be executed prior to the time the FERR# signal is reset by the Intel OverDrive Processor. This function is implemented to allow exact compatibility with the AT implementation. Most programs reinitialize the floating point unit before continuing after an error is detected. The floating point unit can be reinitialized using one of the following four instructions: FCLEX, FINIT, FSAVE, FSTENV.

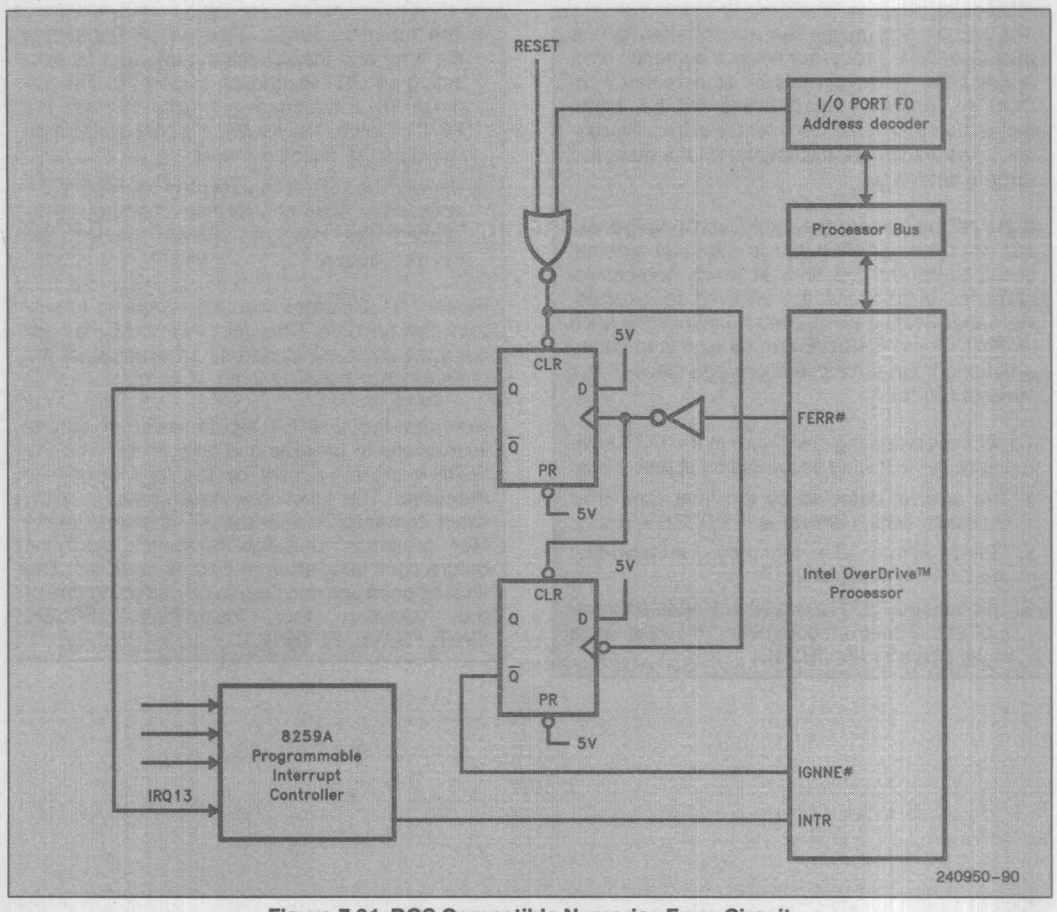


Figure 7.31. DOS Compatible Numerics Error Circuit

8.0 TESTABILITY

Testing in the Intel486 SX microprocessor/Intel OverDrive Processor can be divided into two categories: Built-in Self Test (BIST) and external testing. The BIST tests the non-random logic, control ROM (CROM), translation lookaside buffer (TLB) and on-chip cache memory. External tests can be run on the TLB and the on-chip cache. The Intel486 SX microprocessor/Intel OverDrive Processor also has a test mode in which all outputs are tristated.

8.1 Built-In Self Test (BIST)

The BIST is initiated by asserting AHOLD (address hold) on the falling edge of RESET. AHOLD is a synchronous signal only. It should be asserted in the clock prior to RESET going from High to Low to start BIST. FLUSH# must also be asserted (driven low) prior to the falling edge of RESET to start BIST. FLUSH# must be deasserted (driven high) during BIST. A20M# must be deasserted (driven high) during the falling edge of RESET to start BIST. The BIST takes approximately 2^{20} clocks, or approximately 54 milliseconds with a 20 MHz Intel486 SX microprocessor/Intel487 SX Math CoProcessor. The BIST takes approximately 600 thousand bus clocks, or approximately 24 milliseconds, with a 25 MHz OverDrive Processor. No bus cycles will be run by the Intel486 SX microprocessor/Intel OverDrive Processor until the BIST is concluded. Note that for the Intel486 SX microprocessor/Intel OverDrive Processor the RESET must be active for 15 clocks with or without BIST being enabled for warm resets.

The results of BIST is stored in the EAX register. The Intel486 SX microprocessor/Intel OverDrive Processor has successfully passed the BIST if the contents of the EAX register are zero. If the results in EAX are not zero then the BIST has detected a flaw in the Intel486 SX microprocessor/Intel OverDrive Processor. The Intel486 SX microprocessor/Intel OverDrive Processor performs reset and begins normal operation at the completion of the BIST.

The non-random logic, control ROM, on-chip cache and translation lookaside buffer (TLB) are tested during the BIST.

The cache portion of the BIST verifies that the cache is functional and that it is possible to read and write to the cache. The BIST manipulates test registers TR3, TR4 and TR5 while testing the cache. These test registers are described in Section 8.2.

The cache testing algorithm writes a value to each cache entry, reads the value back, and checks that the correct value was read back. The algorithm may be repeated more than once for each of the 512 cache entries using different constants.

The TLB portion of the BIST verifies that the TLB is functional and that it is possible to read and write to the TLB. The BIST manipulates test registers TR6 and TR7 while testing the TLB. TR6 and TR7 are described in Section 8.3.

8.2 On-Chip Cache Testing

The on-chip cache testability hooks are designed to be accessible during the BIST and for assembly language testing of the cache.

The Intel486 SX microprocessor/Intel OverDrive Processor contains a cache fill buffer and a cache read buffer. For testability writes, data must be written to the cache fill buffer before it can be written to a location in the cache. Data must be read from a cache location into the cache read buffer before the microprocessor can access the data. The cache fill and cache read buffer are both 128 bits wide.

8.2.1 CACHE TESTING REGISTERS TR3, TR4 AND TR5

Figure 8.1 shows the three cache testing registers: the Cache Data Test Register (TR3), the Cache Status Test Register (TR4) and the Cache Control Test Register (TR5). External access to these registers is provided through MOV reg,TREG and MOV TREG, reg instructions.

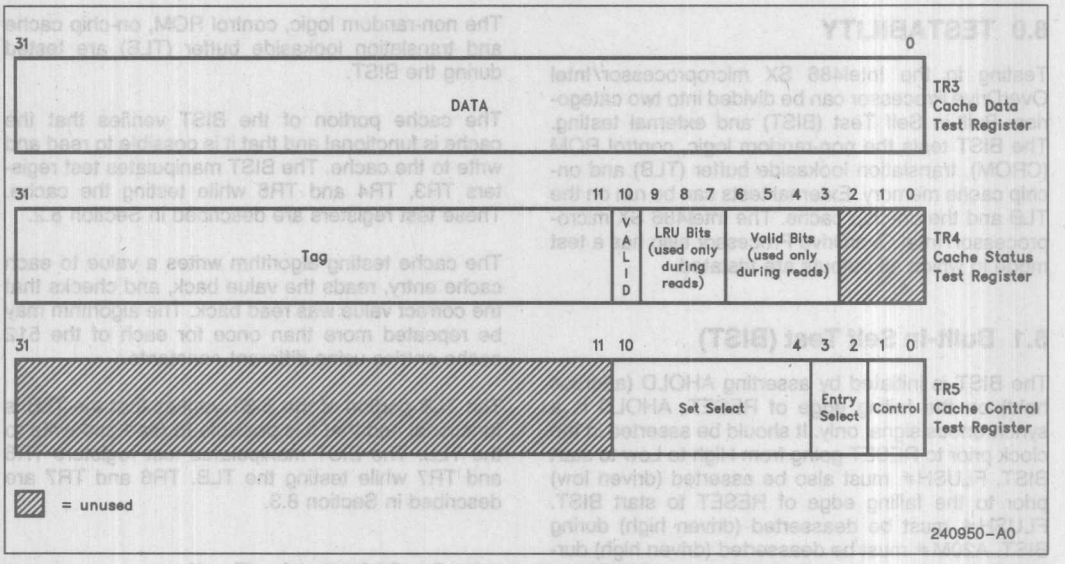


Figure 8.1. Cache Test Registers

Cache Data Test Register: TR3

The cache fill buffer and the cache read buffer can only be accessed through TR3. Data to be written to the cache fill buffer must first be written to TR3. Data read from the cache read buffer must be loaded into TR3.

TR3 is 32 bits wide while the cache fill and read buffers are 128 bits wide. 32 bits of data must be written to TR3 four times to fill the cache fill buffer. 32 bits of data must be read from TR3 four times to empty the cache read buffer. The entry select bits in TR5 determine which 32 bits of data TR3 will access in the buffers.

Cache Status Test Register: TR4

TR4 handles tag, LRU and valid bit information during cache tests. TR4 must be loaded with a tag and a valid bit before a write to the cache. After a read from a cache entry, TR4 contains the tag and valid bit from that entry, and the LRU bits and four valid bits from the accessed set.

Cache Control Test Register: TR5

TR5 specifies which testability operation will be performed and the set and entry within the set which will be accessed.

The seven bit set select field determines which of the 128 sets will be accessed.

The functionality of the two entry select bits depend on the state of the control bits. When the fill or read buffers are being accessed, the entry select bits point to the 32-bit location in the buffer being accessed. When a cache location is specified, the entry select bits point to one of the four entries in a set. Refer to Table 8.1.

Five testability functions can be performed on the cache. The two control bits in TR5 specify the operation to be executed. The five operations are:

1. Write cache fill buffer
2. Perform a cache testability write
3. Perform a cache testability read
4. Read the cache read buffer
5. Perform a cache flush

Table 8.1 shows the encoding of the two control bits in TR5 for the cache testability functions. Table 8.1 also shows the functionality of the entry and set select bits for each control operation.

The cache tests attempt to use as much of the normal operating circuitry as possible. Therefore when cache tests are being performed, the cache must be disabled (the CD and NW bits in control register must be set to 1 to disable the cache. See Section 5).

8.2.2 CACHE TESTABILITY WRITE

A testability write to the cache is a two step process. First the cache fill buffer must be loaded with 128 bits of data and TR4 loaded with the tag and valid bit. Next the contents of the fill buffer are written to a cache location. Sample assembly code to do a write is given in Figure 8.2.

Table 8.1. Cache Control Bit Encoding and Effect of Control Bits on Entry Select and Set Select Functionality

Control Bits		Operation	Entry Select Bits Function	Set Select Bits
Bit 1	Bit 0			
0	0	Enable { Fill Buffer Write / Read Buffer Read	Select 32-bit location in fill/read buffer	—
0	1	Perform Cache Write	Select an entry in set.	Select a set to write to
1	0	Perform Cache Read	Select an entry in set.	Select a set to read from
1	1	Perform Flush Cache	—	—

Sample Assembly Code

An example assembly language sequence to perform a cache write is:

```
;
; eax. ebx. ecx. edx contain the cache line to write
; edi contains the tag information to load
; CR0 already says to enable reads/write to TR5
;
; fill the cache buffer
        mov esi,0          ; set up command
        mov tr5,esi        ; load to TR5
        mov tr3,eax        ; load data into cache fill buffer
        mov esi,4
        mov tr5,esi
        mov tr3,ebx
        mov esi,8
        mov tr5,esi
        mov tr3,ecx
        mov esi,0ch
        mov tr5,esi
        mov tr3,edx
;
; load the Cache Status Register
;
        mov tr4,edi        ; load 21-bit tag and valid bit
;
; perform the cache write
;
        mov esi,1
        mov tr5,esi        ; write the cache (set 0, entry 0)
```

An example assembly language sequence to perform a cache read is:

```
;
; data into eax, ebx, ecx, edx; status into edi
;
; read the cache line back
;
        mov esi,2
        mov tr5,esi        ; do cache testability read (set 0, entry 0)
;
; read the data from the read buffer
;
        mov esi,0
        mov tr5,esi
        mov eax,tr3
        mov esi,4
        mov tr5,esi
        mov ebx,tr3
        mov esi,8
        mov tr5,esi
        mov ecx,tr3
        mov esi,0ch
        mov tr5,esi
        mov edx,tr3
;
; read the status from TR4
;
        mov edi,tr4
```

Figure 8.2 Sample Assembly Code for Cache Testing

Loading the fill buffer is accomplished by first writing to the entry select bits in TR5 and setting the control bits in TR5 to 00. The entry select bits identify one of four 32-bit locations in the cache fill buffer to put 32 bits of data. Following the write to TR5, TR3 is written with 32 bits of data which are immediately placed in the cache fill buffer. Writing to TR3 initiates the write to the cache fill buffer. The cache fill buffer is loaded with 128 bits of data by writing to TR5 and TR3 four times using a different entry select location each time.

TR4 must be loaded with the 21-bit tag and valid bit (bit 10 in TR4) before the contents of the fill buffer are written to a cache location.

The contents of the cache fill buffer are written to a cache location by writing TR5 with a control field of 01 along with the set select and entry select fields. The set select and entry select field indicate the location in the cache to be written. The normal cache LRU update circuitry updates the internal LRU bits for the selected set.

Note that a cache testability write can only be done when the cache is disabled for replaces (the CD bit is control register 0 is reset to 1). Also note that care must be taken when directly writing to entries in the cache. If the entry is set to overlap an area of memory that is being used in external memory, that cache entry could inadvertently be used instead of the external memory. Of course, this is exactly the type of operation that one would desire if the cache were to be used as a high speed RAM. Also note that a memory reference (or any external bus cycle) should not occur in between the move to TR4 and the move to TR5, in order to avoid having the value in TR4 change due to the memory reference.

8.2.3 CACHE TESTABILITY READ

A cache testability read is a two step process. First the contents of the cache location are read into the cache read buffer. Next the data is examined by reading it out of the read buffer. Sample assembly code to do a testability read is given in Figure 8.2.

Reading the contents of a cache location into the cache read buffer is initiated by writing TR5 with the control bits set to 10 and the desired seven-bit set select and two-bit entry select. In response to the write to TR5, TR4 is loaded with the 21-bit tag field and the single valid bit from the cache entry read. TR4 is also loaded with the three LRU bits and four valid bits corresponding to the cache set that was accessed. The cache read buffer is filled with the 128-bit value which was found in the data array at the specified location.

The contents of the read buffer are examined by performing four reads of TR3. Before reading TR3 the entry select bits in TR5 must loaded to indicate which of the four 32-bit words in the read buffer to transfer into TR3 and the control bits in TR5 must be loaded with 00. The register read of TR3 will initiate the transfer of the 32-bit value from the read buffer to the specified general purpose register.

Note that it is very important that the entire 128-bit quantity from the read buffer and also the information from TR4 be read before any memory references are allowed to occur. If memory operations are allowed to happen, the contents of the read buffer will be corrupted. This is because the testability operations use hardware that is used in normal memory accesses for the Intel486 SX microprocessor/Intel OverDrive Processor whether the cache is enabled or not.

8.2.4 FLUSH CACHE

The control bits in TR5 must be written with 11 to flush the cache. None of the other bits in TR5 have any meaning when 11 is written to the control bits. Flushing the cache will reset the LRU bits and the valid bits to 0, but will not change the cache tag or data arrays.

When the cache is flushed by writing to TR5 the special bus cycle indicating a cache flush to the external system is not run (see Section 7.2.11, Special Bus Cycles). The cache should be flushed with the instruction INVD (Invalidate Data Cache) instruction or the WBINVD (Write-back and Invalidate Data Cache) instruction.

8.3 Translation Lookaside Buffer (TLB) Testing

The Intel486 SX microprocessor/Intel OverDrive Processor TLB testability hooks are similar to those in the Intel386 microprocessor. The testability hooks have been enhanced to provide added test features and to include new features in the Intel486 SX microprocessor/Intel OverDrive Processor. The TLB testability hooks are designed to be accessible during the BIST and for assembly language testing of the TLB.

8.3.1 TRANSLATION LOOKASIDE BUFFER ORGANIZATION

The Intel486 SX microprocessor/Intel OverDrive Processor TLB is 4-way set associative and has space for 32 entries. The TLB is logically split into three blocks shown in Figure 8.3.

The data block is physically split into four arrays, each with space for eight entries. An entry in the data block is 22 bits wide containing a 20-bit physical address and two bits for the page attributes. The page attributes are the PCD (page cache disable) bit and the PWT (page write-through) bit. Refer to Section 4.5.4 for a discussion of the PCD and PWT bits.

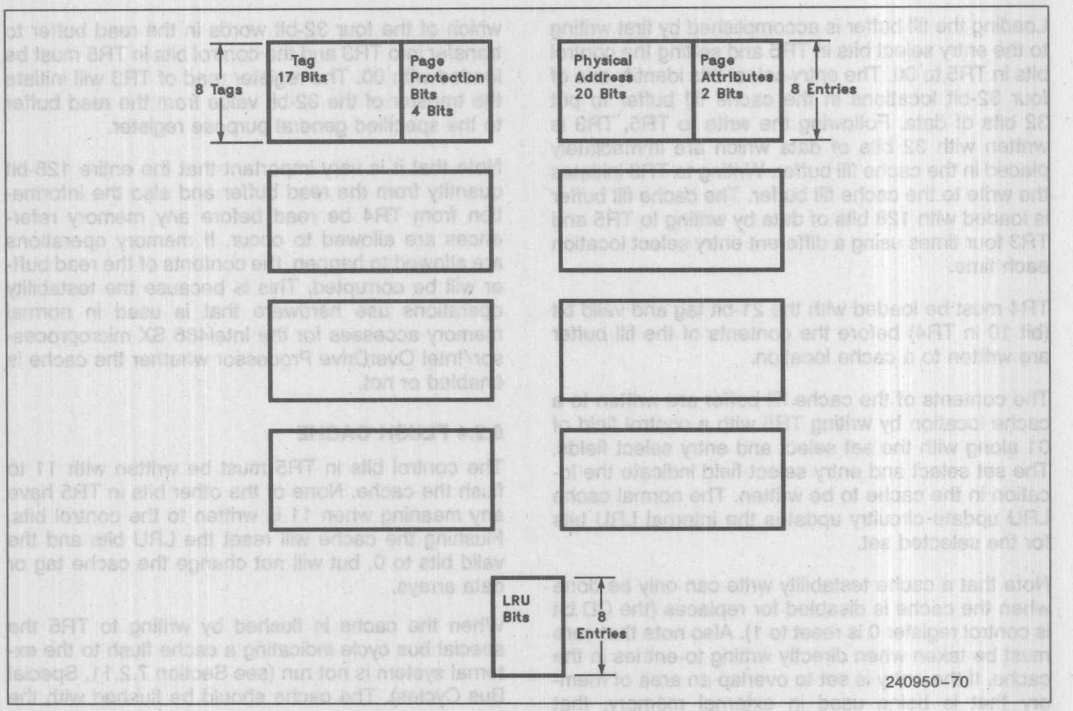


Figure 8.3. TLB Organization

The tag block is also split into four arrays, one for each of the data arrays. A tag entry is 21 bits wide containing a 17-bit linear address and four protection bits. The protection bits are valid (V), user/supervisor (U/S), read/write (R/W) and dirty (D).

The third block contains eight three bit quantities used in the pseudo least recently used (LRU) replacement algorithm. These bits are called the LRU bits. The LRU replacement algorithm used in the TLB is the same as used by the on-chip cache. For a description of this algorithm refer to Section 5.5.

8.3.2 TLB TEST REGISTERS TR6 AND TR7

The two TLB test registers are shown in Figure 8.4. TR6 is the command test register and TR7 is the data test register. External access to these registers is provided through MOV reg,TREG and MOV TREG,reg instructions.

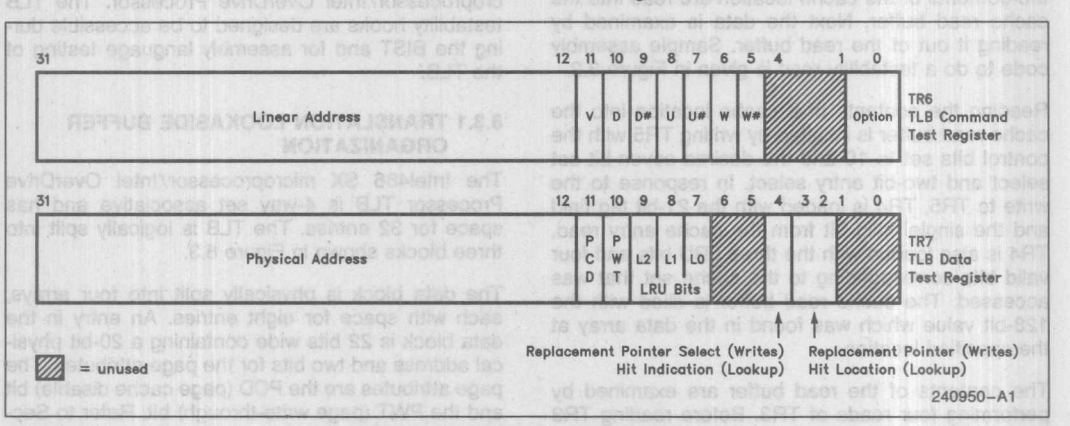


Figure 8.4. TLB Test Registers

Table 8.2. Meaning of a Pair of TR6 Protection Bits

TR6 Protection Bit (B)	TR6 Protection Bit # (B#)	Meaning on TLB Write Operation	Meaning on TLB Lookup Operation
0	0	Undefined	Miss any TLB TAG Bit B
0	1	Write 0 to TLB TAG Bit B	Match TLB TAG Bit B if 0
1	0	Write 1 to TLB TAG Bit B	Match TLB TAG Bit B if 1
1	1	Undefined	Match any TLB TAG Bit B

Command Test Register: TR6

TR6 contains the tag information and control information used in a TLB test. Loading TR6 with tag and control information initiates a TLB write or lookup test.

TR6 contains three bit fields, a 20-bit linear address (bits 12–31), seven bits for the TLB tag protection bits (bits 5-11) and one bit (bit 0) to define the type of operation to be performed on the TLB.

The 20-bit linear address forms the tag information used in the TLB access. The lower three bits of the linear address select which of the eight sets are accessed. The upper 17 bits of the linear address form the tag stored in the tag array.

The seven TLB tag protection bits are described below.

V: The valid bit for this TLB entry

D,D#: The dirty bit for/from the TLB entry

U,U#: The user/supervisor bit for/from the TLB entry

W,W#: The read/write bit for/from the TLB entry

Two bits are used to represent the D, U/S and R/W bits in the TLB tag to permit the option of a forced miss or hit during a TLB lookup operation. The forced miss or hit will occur regardless of the state of the actual bit in the TLB. The meaning of these pairs of bits is given in Table 8.2.

The operation bit in TR6 determines if the TLB test operation will be a write or a lookup. The function of the operation bit is given in Table 8.3.

Table 8.3. TR6 Operation Bit Encoding

TR6 Bit 0	TLB Operation to Be Performed
0	TLB Write
1	TLB Lookup

Data Test Register: TR7

TR7 contains the information stored or read from the data block during a TLB test operation. Before a TLB test write, TR7 contains the physical address and the page attribute bits to be stored in the entry. After a TLB test lookup hit, TR7 contains the physical address, page attributes, LRU bits and entry location from the access.

TR7 contains a 20-bit physical address (bits 12–31), two bits for PCD (bit 11) and PWT (bit 10) and three bits for the LRU bits (bits 7–9). The LRU bits in TR7 are only used during a TLB lookup test. The functionality of TR7 bit 4 differs for TLB writes and lookups. The encoding of bit 4 is defined in Tables 8.4 and 8.5. Finally TR7 contains two bits (bits 2–3) to specify a TLB replacement pointer or the location of a TLB hit.

Table 8.4. Encoding of Bit 4 of TR7 on Writes

TR7 Bit 4	Replacement Pointer Used on TLB Write
0	Pseudo-LRU Replacement Pointer
1	Data Test Register Bits 3:2

Table 8.5. Encoding of Bit 4 of TR7 on Lookups

TR7 Bit 4	Meaning after TLB Lookup Operation
0	TLB Lookup Resulted in a Miss
1	TLB Lookup Resulted in a Hit

A replacement pointer is used during a TLB write. The pointer indicates which of the four entries in an accessed set is to be written. The replacement pointer can be specified to be the internal LRU bits or bits 2–3 in TR7. The source of the replacement pointer is specified by TR7 bit 4. The encoding of bit 4 during a write is given by Table 8.4.

Note that both testability writes and lookups affect the state of the internal LRU bits regardless of the replacement pointer used. All TLB write operations (testability or normal operation) cause the written entry to become the most recently used. For example, during a testability write with the replacement pointer specified by TR7 bits 2–3, the indicated entry is written and that entry becomes the most recently used as specified by the internal LRU bits.

There are two TLB testing operations: write entries into the TLB, and perform TLB lookups. One major enhancement over TLB testing in the Intel386 microprocessor is that paging need not be disabled while executing testability writes or lookups.

Note that any time one TLB set contains the same linear address in more than one of its entries, looking up that linear address will give unpredictable results. Therefore a single linear address should not be written to one TLB set more than once.

8.3.3 TLB WRITE TEST

To perform a TLB write TR7 must be loaded followed by a TR6 load. The register operations must be performed in this order since the TLB operation is triggered by the write to TR6.

TR7 is loaded with a 20-bit physical address and values for PCD and PWT to be written to the data portion of the TLB. In addition, bit 4 of TR7 must be loaded to indicate whether to use TR7 bits 3-2 or the internal LRU bits as the replacement pointer on the TLB write operation. Note that the LRU bits in TR7 are not used in a write test.

TR6 must be written to initiate the TLB write operation. Bit 0 in TR6 must be reset to zero to indicate a TLB write. The 20-bit linear address and the seven page protection bits must also be written in TR6 to specify the tag portion of the TLB entry. Note that the three least significant bits of the linear address specify which of the eight sets in the data block will be loaded with the physical address data. Thus only 17 of the linear address bits are stored in the tag array.

8.3.4 TLB LOOKUP TEST

To perform a TLB lookup it is only necessary to write the proper tags and control information into TR6. Bit 0 in TR6 must be set to 1 to indicate a TLB lookup. TR6 must be loaded with a 20-bit linear address and the seven protection bits. To force misses and matches of the individual protection bits on TLB lookups, set the seven protection bits as specified in Table 8.2.

A TLB lookup operation is initiated by the write to TR6. TR7 will indicate the result of the lookup operation following the write to TR6. The hit/miss indication can be found in TR7 bit 4 (see Table 8.5).

TR7 will contain the following information if bit 4 indicated that the lookup test resulted in a hit. Bits 2–3 will indicate in which set the match occurred. The 22 most significant bits in TR7 will contain the physical address and page attributes contained in the entry. Bits 9–7 will contain the LRU bits associated with the accessed set. The state of the LRU bits is previous to their being updated for the current lookup.

If bit 4 in TR7 indicated that the lookup test resulted in a miss the remaining bits in TR7 are undefined.

Again it should be noted that a TLB testability lookup operation affects the state of the LRU bits. The LRU bits will be updated if a hit occurred. The entry which was hit will become the most recently used.

8.4 Tri-State Output Test Mode

The Intel486 SX microprocessor/Intel OverDrive Processor provides the ability to float all its outputs and bidirectional pins. This includes all pins floated during bus hold as well as pins which are never floated in normal operation of the chip (HLDA, BREQ, FERR# and PCHK#). When the Intel486 SX microprocessor/Intel OverDrive Processor is in the tri-state output test mode external testing can be used to test board connections.

The tri-state test mode is invoked if FLUSH# is sampled active at the falling edge of RESET. FLUSH# is an asynchronous signal. When driven asynchronously, FLUSH# should be asserted for 2 clocks before and 2 clocks after RESET is deasserted. If FLUSH# is driven synchronously, the tri-state output test mode is initiated by driving FLUSH# so that it is sampled active in the clock prior to RESET going low and ensuring that specified setup and hold times are met. The outputs are guaranteed to tri-state no later than 10 clocks after RESET goes low (see Figure 6.4). The Intel486 SX microprocessor/Intel OverDrive Processor remains in the tri-state test mode until the next RESET.

8.5 Intel486™ SX CPU Microprocessor Boundary Scan (JTAG)

The Intel486 SX CPU (PQFP Version Only) provides additional testability features compatible with the IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std. 1149.1). The test logic provided allows for testing to insure that components function correctly, that interconnections between various components are correct, and that various components interact correctly on the printed circuit board.

The boundary scan test logic consists of a boundary scan register and support logic that are accessed through a test access port (TAP). The TAP provides a simple serial interface that makes it possible to test all signal traces with only a few probes.

The TAP can be controlled via a bus master. The bus master can be either automatic test equipment or a component (PLD) that interfaces to the four-pin test bus.

8.5.1 BOUNDARY SCAN ARCHITECTURE

The boundary scan test logic contains the following elements:

— Test access port (TAP), consisting of input pins TMS, TCK, and TDI; and output pin TDO.

— TAP controller, which interprets the inputs on the test mode select (TMS) line and performs the corresponding operation. The operations performed by the TAP include controlling the instruction and data registers within the component.

— Instruction register (IR), which accepts instruction codes shifted into the test logic on the test data input (TDI) pin. The instruction codes are used to select the specific test operation to be performed or the test data register to be accessed.

— Test data registers: The Intel486 SX CPU contains three test data registers: Bypass register (BPR), Device Identification register (DID), and Boundary Scan register (BSR).

The instruction and test data registers are separate shift-register paths connected in parallel and have a common serial data input and a common serial data output connected to the TAP signals, TDI and TDO, respectively.

8.5.2 DATA REGISTERS

The Intel486 SX CPU contains the two required test data registers; bypass register and boundary scan register. In addition, they also have a device identification register.

Each test data register is serially connected to TDI and TDO, with TDI connected to the most significant bit and TDO connected to the least significant bit of the test data register.

Data is shifted one stage (bit position within the register) on each rising edge of the test clock (TCK).

In addition the Intel486 SX CPU contains a runbist register to support the RUNBIST boundary scan instruction.

8.5.2.1 Bypass Register

The Bypass Register is a one-bit shift register that provides the minimal length path between TDI and TDO. This path can be selected when no test operation is being performed by the component to allow rapid movement of test data to and from other components on the board. While the bypass register is selected data is transferred from TDI to TDO without inversion.

8.5.2.2 Boundary Scan Register

The Boundary Scan Register is a single shift register path containing the boundary scan cells that are connected to all input and output pins of the Intel486 SX CPU. Figure 8.5 shows the logical structure of the boundary scan register. While output cells determine the value of the signal driven on the corresponding pin, input cells only capture data; they do not affect the normal operation of the device. Data is transferred without inversion from TDI to TDO through the boundary scan register during scanning. The boundary scan register can be operated by the EXTEST and SAMPLE instructions. The boundary scan register order is described in Section 8.5.5.

8.5.2.3 Device Identification Register

The Device Identification Register contains the manufacturer's identification code, part number code, and version code in the format shown in Figure 8.6. Table 8.6 lists the codes corresponding to the Intel486 SX CPU.

8.5.2.4 Runbist Register

The Runbist Register is a one bit register used to report the results of the Intel486 SX CPU BIST when it is initiated by the RUNBIST instruction. This register is loaded with a "1" prior to invoking the BIST and is loaded with "0" upon successful completion.

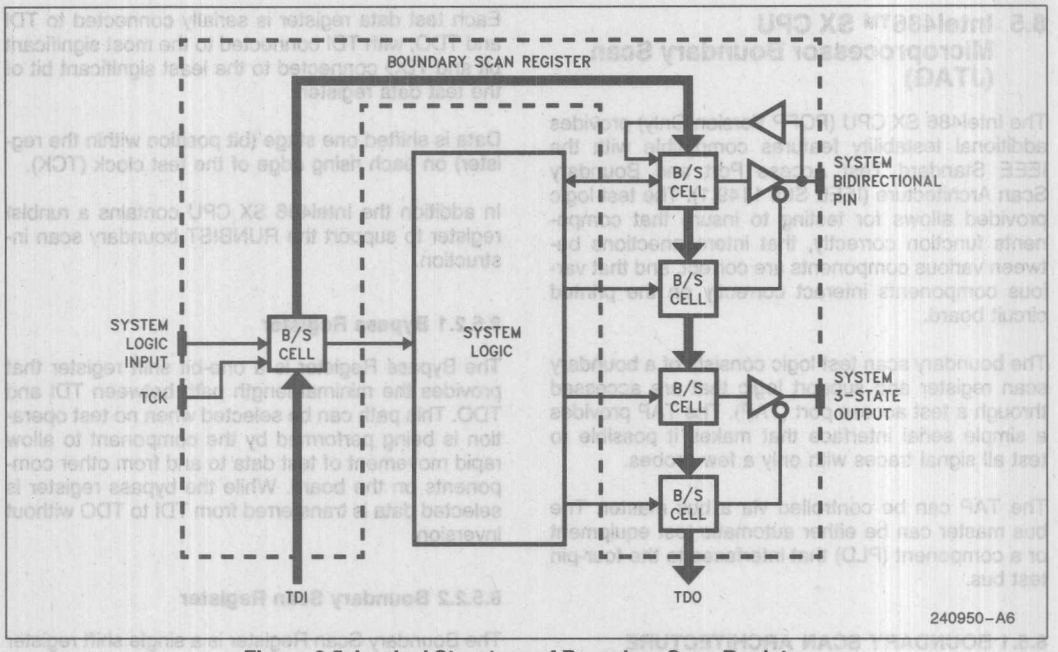


Figure 8.5. Logical Structure of Boundary Scan Register

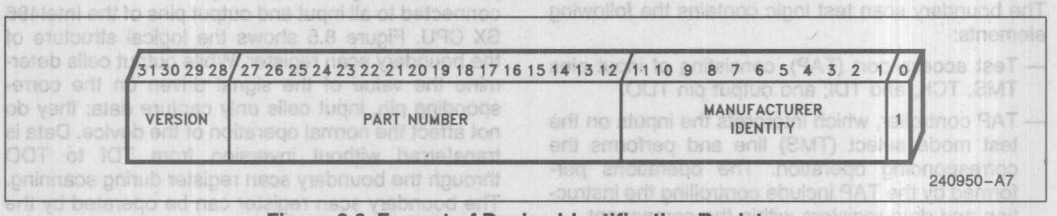


Figure 8.6. Format of Device Identification Register

Table 8.6

Component Code/Step	Device ID (Boundary Scan)
Intel486 SX	
(cA)	00427013h
(B)	20282013h
(cB)	80282013h

8.5.3 INSTRUCTION REGISTER

The Instruction Register (IR) allows instructions to be serially shifted into the device. The instruction selects the particular test to be performed, the test data register to be accessed, or both. The instruction register is four (4) bits wide. The most significant bit is connected to TDI and the least significant bit is connected to TDO. There are no parity bits associated with the Instruction register. Upon entering the Capture-IR TAP controller state, the Instruction register is loaded with the default instruction "0001", SAMPLE/PRELOAD. Instructions are shifted into the instruction register on the rising edge of TCK while the TAP controller is in the SHIFT-IR state.

8.5.3.1 Intel486™ SX CPU Boundary Scan Instruction Set

The Intel486 SX CPU supports all three mandatory boundary scan instructions (BYPASS, SAMPLE/PRELOAD, and EXTEST) along with two optional instructions (IDCODE and RUNBIST). Table 8.7 lists the Intel486 SX CPU boundary scan instruction codes. The instructions listed as PRIVATE cause TDO to become enabled in the Shift-DR state and cause "0" to be shifted out of TDO on the rising edge of TCK. Execution of the PRIVATE instructions will not cause hazardous operation of the Intel486 SX CPU.

EXTEST The instruction code is "0000". The EXTEST instruction allows testing of circuitry external to the component package, typically board interconnects. It does so by driving the values loaded into the Intel486 SX CPU's boundary scan register out on the output pins corresponding to each boundary scan cell and capturing the values on Intel486 SX CPU input pins to be loaded into their corresponding boundary scan register locations. I/O pins are selected as input or output, depending on the value loaded into their control setting locations in the boundary scan register. Values shifted into input latches in the boundary scan register are never used by the internal logic of the Intel486 SX CPU.

NOTE:
After using the EXTEST instruction, the Intel486 SX CPU must be reset before normal (non-boundary scan) use.

SAMPLE/PRELOAD The instruction code is "0001". The SAMPLE/PRELOAD has two functions that it performs. When the TAP controller is in the Capture-DR state, the SAMPLE/PRELOAD instruction allows a "snap-shot" of the normal operation of the component without interfering with that normal operation. The instruction causes boundary scan register cells associated with outputs to sample the value being driven by the Intel486 SX CPU. It causes the cells associated with inputs to sample the value being driven into the Intel486 SX CPU. On both outputs and inputs the sampling occurs on the rising edge of TCK. When the TAP controller is in the Update-DR state, the SAMPLE/PRELOAD instruction preloads data to the device pins to be driven to the board by executing the EXTEST instruction. Data is preloaded to the pins from the boundary scan register on the falling edge of TCK.

IDCODE The instruction code is "0010". The IDCODE instruction selects the device identification register to be connected to TDI and TDO, allowing the device identification code to be shifted out of the device on TDO. Note that the device identification register is not altered by data being shifted in on TDI.

BYPASS The instruction code is "1111". The BYPASS instruction selects the bypass register to be connected to TDI and TDO, effectively bypassing the test logic on the Intel486 SX CPU by reducing the shift length of the device to one bit. Note that an open circuit fault in the board level test data path will cause the bypass register to be selected following an instruction scan cycle due to the pull-up resistor on the TDI input. This has been done to prevent any unwanted interference with the proper operation of the system logic.

RUNBIST The instruction code is "1000". The RUNBIST instruction selects the one (1) bit runbist register, loads a value of "1" into the runbist register, and connects it to TDO. It also initiates the built-in self test (BIST) feature of the Intel486 SX CPU, which is able to detect approximately 60% of the stuck-at faults on the Intel486 SX CPU. The Intel486 SX CPU ac/dc specifications for V_{CC} and CLK must be met and RESET must have been asserted at least once prior to executing the RUNBIST boundary scan instruction. After loading the RUNBIST instruction code in the instruction register, the TAP controller must be placed in the Run-Test/Idle state. BIST begins on the first rising edge of TCK after entering the Run-Test/Idle state. The TAP controller must remain in the Run-Test/Idle state until BIST is completed. It

Table 8.7

Instruction Code	Instruction Name	Instruction Code	Instruction Name
0000	EXTEST	1000	RUNBIST
0001	SAMPLE	1001	PRIVATE
0010	IDCODE	1010	PRIVATE
0011	PRIVATE	1011	PRIVATE
0100	PRIVATE	1100	PRIVATE
0101	PRIVATE	1101	PRIVATE
0110	PRIVATE	1110	PRIVATE
0111	PRIVATE	1111	BYPASS

requires 1.2 million clock (CLK) cycles to complete BIST and report the result to the runbist register. After completing the 1.2 million clock (CLK) cycles, the value in the runbist register should be shifted out on TDO during the Shift-DR state. A value of "0" being shifted out on TDO indicates BIST successfully completed. A value of "1" indicates a failure occurred. After executing the RUNBIST instruction, the Intel486 SX CPU must be reset prior to normal operation.

8.5.4 TEST ACCESS PORT (TAP) CONTROLLER

The TAP controller is a synchronous, finite state machine. It controls the sequence of operations of the test logic. The TAP controller changes state only in response to the following events:

1. a rising edge of TCK
2. power-up.

The value of the test mode state (TMS) input signal at a rising edge of TCK controls the sequence of the state changes. The state diagram for the TAP controller is shown in Figure 8.7. Test designers must consider the operation of the state machine in order to design the correct sequence of values to drive on TMS.

8.5.4.1 Test-Logic-Reset State

In this state, the test logic is disabled so that normal operation of the device can continue unhindered. This is achieved by initializing the instruction register such that the IDCODE instruction is loaded. No matter what the original state of the controller, the controller enters Test-Logic-Reset state when the TMS input is held high (1) for at least five rising edges of TCK. The controller remains in this state while TMS is high. The TAP controller is also forced to enter this state at power-up.

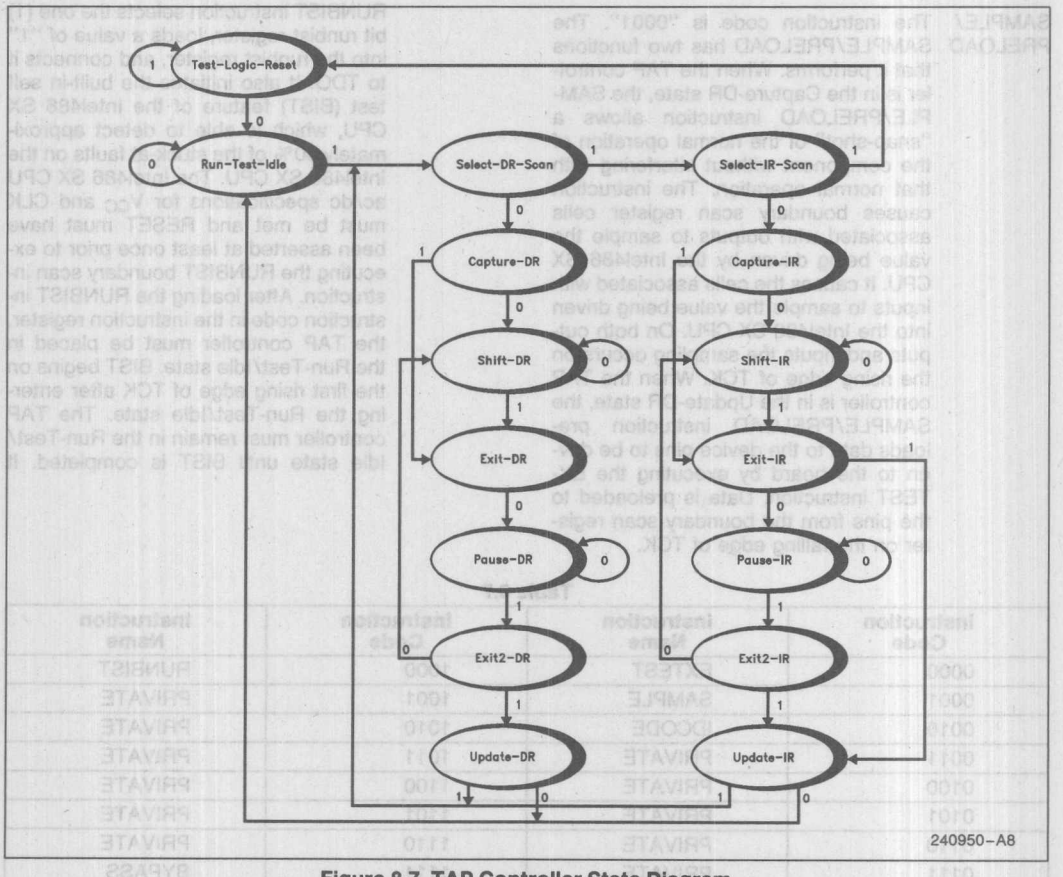


Figure 8.7. TAP Controller State Diagram

8.5.4.2 Run-Test/Idle State

A controller state between scan operations. Once in this state, the controller remains in this state as long as TMS is held low. In devices supporting the RUNBIST instruction, the BIST is performed during this state and the result is reported in the runbist register. For instruction not causing functions to execute during this state, no activity occurs in the test logic. The instruction register and all test data registers retain their previous state. When TMS is high and a rising edge is applied to TCK, the controller moves to the Select-DR state.

8.5.4.3 Select-DR-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held low and a rising edge is applied to TCK when in this state, the controller moves into the Capture-DR state, and a scan sequence for the selected test data register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves to the Select-IR-Scan state.

The instruction does not change in this state.

8.5.4.4 Capture-DR State

In this state, the boundary scan register captures input pin data if the current instruction is EXTEST or SAMPLE/PRELOAD. The other test data registers, which do not have parallel input, are not changed.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-DR state if TMS is high or the Shift-DR state if TMS is low.

8.5.4.5 Shift-DR State

In this controller state, the test data register connected between TDI and TDO as a result of the current instruction, shifts data one stage toward its serial output on each rising edge of TCK.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-DR state if TMS is high or remains in the Shift-DR state if TMS is low.

8.5.4.6 Exit1-DR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-DR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.7 Pause-DR State

The pause state allows the test controller to temporarily halt the shifting of data through the test data register in the serial path between TDI and TDO. An example of using this state could be to allow a tester to reload its pin memory from disk during application of a long test sequence.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves to the Exit2-DR state.

8.5.4.8 Exit2-DR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Shift-DR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.9 Update-DR State

The boundary scan register is provided with a latched parallel output to prevent changes at the parallel output while data is shifted in response to the EXTEST and SAMPLE/PRELOAD instructions. When the TAP controller is in this state and the boundary scan register is selected, data is latched onto the parallel output of this register from the shift-register path on the falling edge of TCK. The data held at the latched parallel output does not change other than in this state.

All shift-register stages in test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.10 Select-IR-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held low and a rising edge is applied to TCK when in this state, the controller moves into the Capture-IR state, and a scan sequence for the instruction register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves to the Test-Logic-Reset state.

The instruction does not change in this state.

8.5.4.11 Capture-IR State

In this controller state the shift register contained in the instruction register loads the fixed value "0001" on the rising edge of TCK.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state. When the controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-IR state if TMS is held high, or the Shift-IR state if TMS is held low.

8.5.4.12 Shift-IR State

In this state the shift register contained in the instruction register is connected between TDI and TDO and shifts data one stage towards its serial output on each rising edge of TCK.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-IR state if TMS is held high, or remains in the Shift-IR state if TMS is held low.

8.5.4.13 Exit1-IR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-IR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.14 Pause-IR State

The pause state allows the test controller to temporarily halt the shifting of data through the instruction register.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves to the Exit2-IR state.

8.5.4.15 Exit2-IR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Shift-IR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.16 Update-IR State

The instruction shifted into the instruction register is latched onto the parallel output from the shift-register path on the falling edge of TCK. Once the new instruction has been latched, it becomes the current instruction.

Test data registers selected by the current instruction retain the previous value.

8.5.5 BOUNDARY SCAN REGISTER CELL

The boundary scan register contains a cell for each pin, as well as cells for control of I/O and tri-state pins.

The following is the bit order of the Intel486 SX CPU boundary scan register: (from left to right and top to bottom)

TDI → WRCTL ABUSCTL BUSCTL MISCCTL ADS# BLAST# PLOCK# LOCK# PCHK# BRDY# BOFF# BS16# BS8# RDY# KEN# HOLD AHOLD CLK HLDA WR# BREQ BE0# BE1# BE2# BE3# MIO# DC# PWT PCD EADS# A20M# RESET FLUSH# INTR NMI Reserved Reserved D31 D30 D29 D28 D27 D26 D25 D24 DP3 D23 D22 D21 D20 D19 D18 D17 D16 DP2 D15 D14 D13 D12 D11 D10 D9 D8 DP1 D7 D6 D5 D4 D3 D2 D1 D0 DP0 A31 A30 A29 A28 A27 A26 A25 A24 A23 A22 A21 A20 A19 A18 A17 A16 A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 UP# A5 A4 A3 A2 → TDO

"Reserved" corresponds to no connect "NC" signals on the Intel486 SX CPU.

All the *CTL cells are control cells that are used to select the direction of bidirectional pins or tristate output pins. If "1" is loaded into the control cell (*CTL), the associated pin(s) are tristated or selected as input. The following lists the control cells and their corresponding pins.

1. WRCTL controls the D31–D0 and DP3–DP0 pins.
2. ABUSCTL controls the A31–A2 pins.
3. BUSCTL controls the ADS#, BLAST#, PLOCK#, LOCK#, WR#, BE0#, BE1#, BE2#, BE3#, MIO#, DC#, PWT, and PCD pins.
4. MISCCTL controls the PCHK#, HLDA, and BREQ pins.

8.5.6 TAP CONTROLLER INITIALIZATION

The TAP controller is automatically initialized when a device is powered up. In addition, the TAP controller can be initialized by applying a high signal level on the TMS input for five TCK periods.

2

8.5.7 BOUNDARY SCAN DESCRIPTION LANGUAGE (BSDL) FILES

Available through Intel.

9.0 DEBUGGING SUPPORT

The Intel486 SX microprocessor/Intel OverDrive Processor provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

1. the code execution breakpoint opcode (0CCH),
2. the single-step capability provided by the TF bit in the flag register, and
3. the code and data breakpoint capability provided by the Debug Registers DR0–3, DR6, and DR7.

9.1 Breakpoint Instruction

A single-byte-opcode breakpoint instruction is available for use by software debuggers. The breakpoint opcode is 0CCH, and generates an exception 3 trap when executed. In typical use, a debugger program can "plant" the breakpoint instruction at all desired code execution breakpoints. The single-byte breakpoint opcode is an alias for the two-byte general software interrupt instruction, INT n, where n=3. The only difference between INT 3 (0CCh) and INT n is that INT 3 is never IOPL-sensitive but INT n is IOPL-sensitive in Protected Mode and Virtual 8086 Mode.

9.2 Single-Step Trap

If the single-step flag (TF, bit 8) in the EFLAG register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto vectored to exception number 1. Precisely, exception 1 occurs as a trap after the instruction following the instruction which set TF. In typical practice, a debugger sets the TF bit of a flag register image on the debugger's stack. It then typically transfers control to the user program and loads the flag image with a signal instruction, the IRET instruction. The single-step trap occurs after executing one instruction of the user program.

Since the exception 1 occurs as a trap (that is, it occurs after the instruction has already executed), the CS:EIP pushed onto the debugger's stack points to the next unexecuted instruction of the program being debugged. An exception 1 handler, merely by ending with an IRET instruction, can therefore efficiently support single-stepping through a user program.

9.3 Debug Registers

The Debug Registers are an advanced debugging feature of the Intel486 SX microprocessor/Intel OverDrive Processor. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution break-point can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT3 breakpoint opcode.

The Intel486 SX microprocessor/Intel OverDrive Processor contains six Debug Registers, providing the ability to specify up to four distinct breakpoints addresses, breakpoint control options, and read breakpoint status. Initially after reset, breakpoints are in the disabled state. Therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are autovectored to exception number 1.

9.3.1 LINEAR ADDRESS BREAKPOINT REGISTERS (DR0–DR3)

Up to four breakpoint addresses can be specified by writing into Debug Registers DR0–DR3, shown in Figure 9.1. The breakpoint addresses specified are 32-bit linear addresses. Intel486 SX microprocessor/Intel OverDrive Processor hardware continuously compares the linear breakpoint addresses in DR0–DR3 with the linear addresses generated by executing software (a linear address is the result of computing the effective address and adding the 32-bit segment base address). Note that if paging is not enabled the linear address equals the physical address. If paging is enabled, the linear address is translated to a physical 32-bit address by the on-chip paging unit. Regardless of whether paging is enabled or not, however, the breakpoint registers hold linear addresses.

9.3.2 DEBUG CONTROL REGISTER (DR7)

A Debug Control Register, DR7 shown in Figure 9.1, allows several debug control functions such as enabling the breakpoints and setting up other control options for the breakpoints. The fields within the Debug Control Register, DR7, are as follows:

LENi (breakpoint length specification bits)

A 2-bit LEN field exists for each of the four breakpoints. LEN specifies the length of the associated breakpoint field. The choices for data breakpoints are: 1 byte, 2 bytes, and 4 bytes. Instruction execution breakpoints must have a length of 1 (LENi = 00). Encoding of the LENi field is as follows:

LENi Encoding	Breakpoint Field Width	Usage of Least Significant Bits in Breakpoint Address Register i, (i=0–3)
00	1 byte	All 32-bits used to specify a single-byte breakpoint field.
01	2 bytes	A1–A31 used to specify a two-byte, word-aligned breakpoint field. A0 in Breakpoint Address Register is not used.
10	Undefined—do not use this encoding	
11	4 bytes	A2–A31 used to specify a four-byte, dword-aligned breakpoint field. A0 and A1 in Breakpoint Address Register are not used.

The LENi field controls the size of breakpoint field i by controlling whether all low-order linear address bits in the breakpoint address register are used to detect the breakpoint event. Therefore, all breakpoint fields are aligned; 2-byte breakpoint fields begin on Word boundaries, and 4-byte breakpoint fields begin on Dword boundaries.

The following is an example of various size breakpoint fields. Assume the breakpoint linear address in DR2 is 00000005H. In that situation, the following

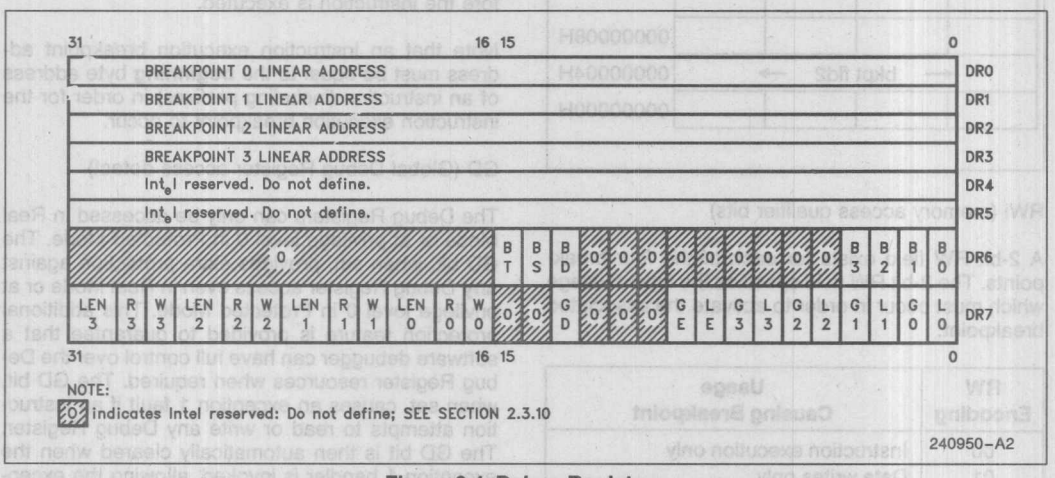
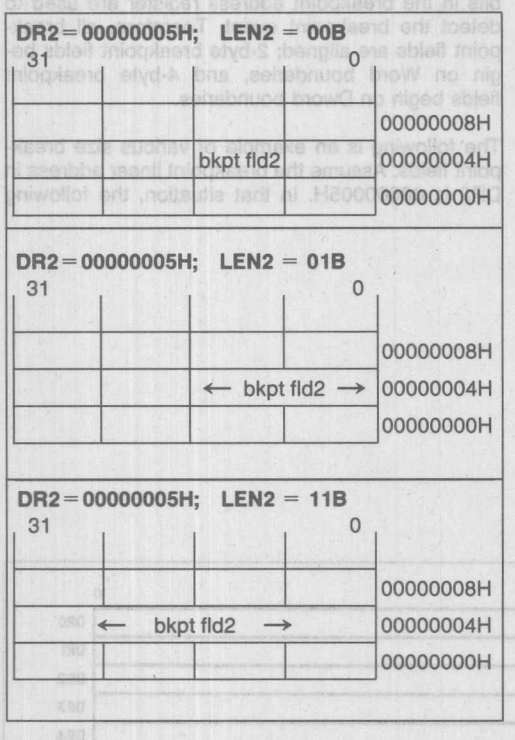


Figure 9.1. Debug Registers

illustration indicates the region of the breakpoint field for lengths of 1, 2, or 4 bytes.



RWi (memory access qualifier bits)

A 2-bit RW field exists for each of the four breakpoints. The 2-bit RW field specifies the type of usage which must occur in order to activate the associated breakpoint.

RW Encoding	Usage Causing Breakpoint
00	Instruction execution only
01	Data writes only
10	Undefined—do not use this encoding
11	Data reads and writes only

RW encoding 00 is used to set up an instruction execution breakpoint. RW encodings 01 or 11 are used to set up write-only or read/write data breakpoints.

Note that **instruction execution breakpoints are taken as faults** (i.e., before the instruction executes), but **data breakpoints are taken as traps** (i.e., after the data transfer takes place).

Using LENi and RWi to Set Data Breakpoint i

A data breakpoint can be set up by writing the linear address into DRi (i = 0–3). For data breakpoints, RWi can = 01 (write-only) or 11 (write/read). LEN can = 00, 01, or 11.

If a data access entirely or partly falls within the data breakpoint field, the data breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 trap will occur.

Using LENi and RWi to Set Instruction Execution Breakpoint i

An instruction execution breakpoint can be set up by writing address of the beginning of the instruction (including prefixes if any) into DRi (i = 0–3). RWi must = 00 and LEN must = 00 for instruction execution breakpoints.

If the instruction beginning at the breakpoint address is about to be executed, the instruction execution breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 fault will occur before the instruction is executed.

Note that an instruction execution breakpoint address must be equal to the **beginning** byte address of an instruction (including prefixes) in order for the instruction execution breakpoint to occur.

GD (Global Debug Register access detect)

The Debug Registers can only be accessed in Real Mode or at privilege level 0 in Protected Mode. The GD bit, when set, provides extra protection against **any** Debug Register access even in Real Mode or at privilege level 0 in Protected Mode. This additional protection feature is provided to guarantee that a software debugger can have full control over the Debug Register resources when required. The GD bit, when set, causes an exception 1 fault if an instruction attempts to read or write any Debug Register. The GD bit is then automatically cleared when the exception 1 handler is invoked, allowing the exception 1 handler free access to the debug registers.

GE and LE (Exact data breakpoint match, global and local)

The breakpoint mechanism of the Intel486 SX microprocessor/Intel OverDrive Processor differs from that of the Intel386 microprocessor. The Intel486 SX microprocessor/Intel OverDrive Processor always does exact data breakpoint matching, regardless of GE/LE bit settings. Any data breakpoint trap will be

reported exactly after completion of the instruction that caused the operand transfer. Exact reporting is provided by forcing the Intel486 SX microprocessor/ Intel OverDrive Processor execution unit to wait for completion of data operand transfers before beginning execution of the next instruction.

When the Intel486 SX microprocessor/Intel OverDrive Processor performs a task switch, the LE bit is cleared. Thus, the LE bit supports fast task switching out of tasks, that have enabled the exact data breakpoint match for their task-local breakpoints. The LE bit is cleared by the Intel486 SX microprocessor/ Intel OverDrive Processor during a task switch, to avoid having exact data breakpoint match enabled in the new task. Note that exact data breakpoint match must be re-enabled under software control.

The Intel486 SX microprocessor/Intel OverDrive Processor GE bit is unaffected during a task switch. The GE bit supports exact data breakpoint match that is to remain enabled during all tasks executing in the system.

Note that **instruction execution** breakpoints are always reported exactly.

Gi and Li (breakpoint enable, global and local)

If either Gi or Li is set then the associated breakpoint (as defined by the linear address in DRi, the length in LENi and the usage criteria in RWi) is enabled. If either Gi or Li is set, and the Intel486 SX microprocessor/Intel OverDrive Processor detects the ith breakpoint condition, then the exception 1 handler is invoked.

When the Intel486 SX microprocessor/Intel OverDrive Processor performs a task switch to a new Task State Segment (TSS), all Li bits are cleared. Thus, the Li bits support fast task switching out of tasks that use some task-local breakpoint registers. The Li bits are cleared by the Intel486 SX microprocessor/Intel OverDrive Processor during a task switch, to avoid spurious exceptions in the new task. Note that the breakpoints must be re-enabled under software control.

All Intel486 SX microprocessor/Intel OverDrive Processor Gi bits are unaffected during a task switch. The Gi bits support breakpoints that are active in all tasks executing in the system.

9.3.3 DEBUG STATUS REGISTER (DR6)

A Debug Status Register, DR6 shown in Figure 9.1, allows the exception 1 handler to easily determine why it was invoked. Note the exception 1 handler can be invoked as a result of one of several events:

1) DR0 Breakpoint fault/trap.
2) DR1 Breakpoint fault/trap.
3) DR2 Breakpoint fault/trap.
4) DR3 Breakpoint fault/trap.
5) Single-step (TF) trap.
6) Task switch trap.
7) Fault due to attempted debug register access when GD = 1.

The Debug Status Register contains single-bit flags for each of the possible events invoking exception 1. Note below that some of these events are faults (exception taken before the instruction is executed), while other events are traps (exception taken after the debug events occurred).

The flags in DR6 are set by the hardware but never cleared by hardware. Exception 1 handler software should clear DR6 before returning to the user program to avoid future confusion in identifying the source of exception 1.

The fields within the Debug Status Register, DR6, are as follows:

Bi (debug fault/trap due to breakpoint 0–3)

Four breakpoint indicator flags, B0–B3, correspond one-to-one with the breakpoint registers in DR0–DR3. A flag Bi is set when the condition described by DRi, LENi, and RWi occurs.

If Gi or Li is set, and if the ith breakpoint is detected, the Intel486 SX microprocessor/Intel OverDrive Processor will invoke the exception 1 handler. The exception is handled as a fault if an instruction execution breakpoint occurred, or as a trap if a data breakpoint occurred.

IMPORTANT NOTE: A flag Bi is set whenever the hardware detects a match condition on **enabled** breakpoint i. Whenever a match is detected on at least one **enabled** breakpoint i, the hardware immediately sets all Bi bits corresponding to breakpoint conditions matching at that instant, whether enabled **or not.** Therefore, the exception 1 handler may see that multiple Bi bits are set, but only set Bi bits corresponding to **enabled** breakpoints (Li or Gi set) are **true** indications of why the exception 1 handler was invoked.

BD (debug fault due to attempted register access when GD bit set)

This bit is set if the exception 1 handler was invoked due to an instruction attempting to read or write to the debug registers when GD bit was set. If such an event occurs, then the GD bit is automatically cleared when the exception 1 handler is invoked, allowing handler access to the debug registers.

BS (debug trap due to single-step)

This bit is set if the exception 1 handler was invoked due to the TF bit in the flag register being set (for single-stepping).

BT (debug trap due to task switch)

This bit is set if the exception 1 handler was invoked due to a task switch occurring to a task having an Intel486 SX microprocessor/Intel OverDrive Processor TSS with the T bit set. Note the task switch into the new task occurs normally, but before the first instruction of the task is executed, the exception 1 handler is invoked. With respect to the task switch operation, the operation is considered to be a trap.

9.3.4 USE OF RESUME FLAG (RF) IN FLAG REGISTER

The Resume Flag (RF) in the flag word can suppress an instruction execution breakpoint when the exception 1 handler returns to a user program at a user address which is also an instruction execution breakpoint.

10.0 INSTRUCTION SET SUMMARY

This section describes the Intel486 SX microprocessor/Intel OverDrive Processor instruction set. Tables 10.1 through 10.3 list all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in Section 10.2, which completely describes the encoding structure and the definition of all fields occurring within the Intel486 SX microprocessor/Intel487 SX Math CoProcessor instructions.

10.1 Intel486™ SX Microprocessor/ Intel487™ SX Math CoProcessor Instruction Encoding and Clock Count Summary

In general, instructions will execute faster on the Intel OverDrive Processor than on the Intel486 SX microprocessor/Intel487 SX Math CoProcessor. The following section describes how to calculate elapsed time for an instruction for the Intel486 SX microprocessor/Intel487 SX Math CoProcessor. See Section 12.3, Instruction Set Summary, for information on calculating elapsed time for instructions executed on the Intel OverDrive Processor.

To calculate elapsed time for an instruction executed on the Intel486 SX microprocessor/Intel487 SX Math CoProcessor, multiply the instruction clock count, as listed in Tables 10.1 through 10.3, by the Intel486 SX microprocessor/Intel487 SX Math CoProcessor clock period (e.g., 50 ns for a 20 MHz Intel486 SX microprocessor/Intel487 SX Math CoProcessor).

For more detailed information on the encodings of instructions, refer to Section 10.2 Instruction Encodings. Section 10.2 explains the general structure of instruction encodings, and defines exactly the encodings of all fields contained within the instruction.

INSTRUCTION CLOCK COUNT ASSUMPTIONS

The Intel486 SX microprocessor/Intel487 SX Math CoProcessor instruction clock count tables give clock counts assuming data and instruction accesses hit in the cache. A separate penalty column defines clocks to add if a data access misses in the cache. The combined instruction and data cache hit rate is over 90%.

A cache miss will force the Intel486 SX microprocessor/Intel OverDrive Processor/Intel487 SX Math CoProcessor to run an external bus cycle. The Intel486 SX microprocessor/Intel OverDrive Processor/Intel487 SX Math CoProcessor 32-bit burst bus is defined as r−b−w.

Where:

r = The number of clocks in the first cycle of a burst read or the number of clocks per data cycle in a non-burst read.

b = The number of clocks for the second and subsequent cycles in a burst read.

w = The number of clocks for a write.

The fastest bus the Intel486 SX microprocessor/Intel OverDrive Processor/Intel487 SX Math CoProcessor can support is 2−1−2 assuming 0 wait states. The clock counts in the cache miss penalty column assume a 2−1−2 bus. For slower busses add r−2 clocks to the cache miss penalty for the first dword accessed. Other factors also affect instruction clock counts.

Instruction Clock Count Assumptions

1. The external bus is available for reads or writes at all times. Else add clocks to reads until the bus is available.
2. Accesses are aligned. Add three clocks to each misaligned access.
3. Cache fills complete before subsequent accesses to the same line. If a read misses the cache during a cache fill due to a previous read or pre-fetch, the read must wait for the cache fill to complete. If a read or write accesses a cache line still being filled, it must wait for the fill to complete.
4. If an effective address is calculated, the base register is not the destination register of the preceding instruction. If the base register is the destination register of the preceding instruction add 1 to the clock counts shown. Back-to-back PUSH and POP instructions are not affected by this rule.
5. An effective address calculation uses one base register and does not use an index register. However, if the effective address calculation uses an index register, 1 clock **may** be added to the clock count shown.
6. The target of a jump is in the cache. If not, add r clocks for accessing the destination instruction of a jump. If the destination instruction is not completely contained in the first dword read, add a maximum of 3b clocks. If the destination instruction is not completely contained in the first 16 byte burst, add a maximum of another r+3b clocks.
7. If no write buffer delay, w clocks are added only in the case in which all write buffers are full. Typically, this case rarely occurs.
8. Displacement and immediate not used together. If displacement and immediate used together, 1 clock **may** be added to the clock count shown.
9. No invalidate cycles. Add a delay of 1 clock for each invalidate cycle if the invalidate cycle contends for the internal cache/external bus when the Intel486 SX microprocessor/Intel OverDrive Processor/Intel487 SX Math CoProcessor needs to use it.
10. Page translation hits in TLB. A TLB miss will add 13, 21 or 28 clocks to the instruction depending on whether the Accessed and/or Dirty bit in neither, one or both of the page entries needs to be set in memory. This assumes that neither page entry is in the data cache and a page fault does not occur on the address translation.
11. No exceptions are detected during instruction execution. Refer to Interrupt Clock Counts Table for extra clocks if an interrupt is detected.
12. Instructions that read multiple consecutive data items (i.e. task switch, POPA, etc.) and miss the cache are assumed to start the first access on a 16-byte boundary. If not, an extra cache line fill may be necessary which may add up to (r+3b) clocks to the cache miss penalty.

Table 10.1. Intel486™ SX Microprocessor/Intel487™ SX Math CoProcessor Integer Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit	Penalty If Cache Miss	Notes
INTEGER OPERATIONS				
MOV = Move:				
reg1 to reg2	1000100W | 11 reg1 reg2	1		
reg2 to reg1	1000101w | 11 reg1 reg2	1		
memory to reg	1000101w | mod reg r/m	1	2	
reg to memory	1000100w | mod reg r/m	1		
Immediate to reg	1100011w | 11000 reg | immediate data	1		
or	1011w reg | immediate data	1		
Immediate to Memory	1100011w | mod 000 r/m | displacement immediate	1		
Memory to Accumulator	1010000w | full displacement	1	2	
Accumulator to Memory	1010001w | full displacement	1		
MOVSX/MOVZX = Move with Sign/Zero Extension				
reg2 to reg1	00001111 | 1011z11w | 11 reg1 reg2	3		
memory to reg	00001111 | 1011z11w | mod reg r/m	3	2	
PUSH = Push				
reg	11111111 | 11 110 reg	4		
or	01010 reg	1		
memory	11111111 | mod 110 r/m	4	1	1
immediate	011010s0 | immediate data	1		
PUSHA = Push All	01100000	11		
POP = Pop				
reg	10001111 | 11 000 reg	4	1	
or	01011 reg	1	2	
memory	10001111 | mod 000 r/m	5	2	1
POPA = Pop All	01100001	9	7/15	16/32
XCHG = Exchange				
reg1 with reg2	1000011w | 11 reg1 reg2	3		2
Accumulator with reg	10010 reg	3		2
Memory with reg	1000011w | mod reg r/m	5		2
NOP = No Operation	10010000	1		
LEA = Load EA to Register	10001101 | mod reg r/m			
no index register		1		
with index register		2		

z	instruction
0	MOVZX
1	MOVSX

Table 10.1. Intel486™ SX Microprocessor/Intel487™ SX Math CoProcessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
Instruction / **TTT**: ADD = Add 000; ADC = Add with Carry 010; AND = Logical AND 100; OR = Logical OR 001; SUB = Subtract 101; SBB = Subtract with Borrow 011; XOR = Logical Exclusive OR 110				
reg1 to reg2	00TTT00w 11 reg1 reg2	1		
reg2 to reg1	00TTT01w 11 reg1 reg2	1		
memory to register	00TTT01w mod reg r/m	2	2	
register to memory	00TTT00w mod reg r/m	3	6/2	U/L
immediate to register	100000sw 11 TTT reg immediate register	1		
immediate to accumulator	00TTT10w immediate data	1		
immediate to memory	100000sw mod TTT r/m immediate data	3	6/2	U/L
Instruction / **TTT**: INC = Increment 000; DEC = Decrement 001				
reg	1111111w 11 TTT reg	1		
or	01TTT reg	1		
memory	1111111w mod TTT r/m	3	6/2	U/L
Instruction / **TTT**: NOT = Logical Complement 010; NEG = Negate 011				
reg	1111011w 11 TTT reg	1		
memory	1111011w mod TTT r/m	3	6/2	U/L
CMP = Compare				
reg1 with reg2	0011100w 11 reg1 reg2	1		
reg2 with reg1	0011101w 11 reg1 reg2	1		
memory with register	0011100w mod reg r/m	2	2	
register with memory	0011101w mod reg r/m	2	2	
immediate with register	100000sw 11 111 reg immediate data	1		
immediate with acc.	0011110w immediate data	1		
immediate with memory	100000sw mod 111 r/m immediate data	2	2	
TEST = Logical Compare				
reg1 and reg2	1000010w 11 reg1 reg2	1		
memory and register	1000010w mod reg r/m	2	2	
immediate and register	1111011w 11 000 reg immediate data	1		
immediate and acc.	1010100w immediate data	1		
immediate and memory	1111011w mod 000 r/m immediate data	2	2	

Table 10.1. Intel486™ SX Microprocessor/Intel487™ SX Math CoProcessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
MUL = Multiply (unsigned)				
acc. with register	1111011w | 11 100 reg			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
acc. with memory	1111011w | mod 100 r/m			
Multiplier-Byte		13/18	1	MN/MX, 3
Word		13/26	1	MN/MX, 3
Dword		13/42	1	MN/MX, 3
IMUL = Integer Multiply (signed)				
acc. with register	1111011w | 11 101 reg			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
acc. with memory	1111011w | mod 101 r/m			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
reg1 with reg2	00001111 | 10101111 | 11 reg1 reg2			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
register with memory	00001111 | 10101111 | mod reg r/m			
Multiplier-Byte		13/18	1	MN/MX, 3
Word		13/26	1	MN/MX, 3
Dword		13/42	1	MN/MX, 3
reg1 with imm. to reg2	011010s1 | 11 reg1 reg2 | immediate data			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
mem. with imm. to reg.	011010s1 | mod reg r/m | immediate data			
Multiplier-Byte		13/18	2	MN/MX, 3
Word		13/26	2	MN/MX, 3
Dword		13/42	2	MN/MX, 3
DIV = Divide (unsigned)				
acc. by register	1111011w | 11 110 reg			
Divisor-Byte		16		
Word		24		
Dword		40		
acc. by memory	1111011w | mod 110 r/m			
Divisor-Byte		16		
Word		24		
Dword		40		
IDIV = Integer Divide (signed)				
acc. by register	1111011w | 11 111 reg			
Divisor-Byte		19		
Word		27		
Dword		43		

2

Table 10.1. Intel486™ SX Microprocessor/Intel487™ SX Math CoProcessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
acc. by memory	1111011w mod 111 r/m			
Divisor-Byte		20		
Word		28		
Dword		44		
CBW = Convert Byte to Word	10011000	3		
CWD = Convert Word to Dword	10011001	3		

Instruction	TTT
ROL = Rotate Left	000
ROR = Rotate Right	001
RCL = Rotate through Carry Left	010
RCR = Rotate through Carry Right	011
SHL/SAL = Shift Logical/Arithmetic Left	100
SHR = Shift Logical Right	101
SAR = Shift Arithmetic Right	111

Not Through Carry (ROL, ROR, SAL, SAR, SHL, and SHR)						
reg by 1	1101000w	11 TTT reg		3		
memory by 1	1101000w	mod TTT r/m		4	6	
reg by CL	1101001w	11 TTT reg		3		
memory by CL	1101001w	mod TTT r/m		4	6	
reg by immediate count	1100000w	11 TTT reg	immediate 8-bit data	2		
mem by immediate count	1100000w	mod TTT r/m	immediate 8-bit data	4	6	
Through Carry (RCL and RCR)						
reg by 1	1101000w	11 TTT reg		3		
memory by 1	1101000w	mod TTT r/m		4	6	
reg by CL	1101001w	11 TTT reg		8/30		MN/MX, 4
memory by CL	1101001w	mod TTT r/m		9/31		MN/MX, 5
reg by immediate count	1100000w	11 TTT reg	immediate 8-bit data	8/30		MN/MX, 4
mem by immediate count	1100000w	mod TTT r/m	immediate 8-bit data	9/31		MN/MX, 5

Instruction	TTT
SHLD = Shift Left Double	100
SHRD = Shift Right Double	101

register with immediate	00001111	10TTT100	11 reg2 reg1	imm 8-bit data	2		
memory by immediate	00001111	10TTT100	mod reg r/m	imm 8-bit data	3	6	
register by CL	00001111	10TTT101	11 reg2 reg1		3		
memory by CL	00001111	10TTT101	mod reg r/m		4	5	
BSWAP = Byte Swap	00001111	11001 reg			1		
XADD = Exchange and Add							
reg1, reg2	00001111	1100000w	11 reg2 reg1		3		
memory, reg	00001111	1100000w	mod reg r/m		4	6/2	U/L
CMPXCHG = Compare and Exchange							
reg1, reg2	00001111	1011000w	11 reg2 reg1		6		
memory, reg	00001111	1011000w	mod reg r/m		7/10	2	6

Table 10.1. Intel486™ SX Microprocessor/Intel487™ SX Math CoProcessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
CONTROL TRANSFER (within segment)				
NOTE: Times are jump taken/not taken				
Jccc = Jump on ccc				
8-bit displacement	0111tttn | 8-bit disp.	3/1		T/NT, 23
full displacement	00001111 | 1000tttn | full displacement	3/1		T/NT, 23
NOTE: Times are jump taken/not taken				
SETcccc = Set Byte on cccc (Times are cccc true/false)				
reg	00001111 | 1001tttn | 11 000 reg	4/3		
memory	00001111 | 1001tttn | mod 000 r/m	3/4		
LOOP = LOOP CX Times	11100010 | 8-bit disp.	7/6		L/NL, 23
LOOPZ/LOOPE = Loop with Zero/Equal	11100001 | 8-bit disp.	9/6		L/NL, 23
LOOPNZ/LOOPNE = Loop while Not Zero	11100000 | 8-bit disp.	9/6		L/NL, 23
JCXZ = Jump on CX Zero	11100011 | 8-bit disp.	8/5		T/NT, 23
JECXZ = Jump on ECX Zero	11100011 | 8-bit disp.	8/5		T/NT, 23
(Address Size Prefix Differentiates JCXZ for JECXZ)				
JMP = Unconditional Jump (within segment)				
Short	11101011 | 8-bit disp.	3		7, 23
Direct	11101001 | full displacement	3		7, 23
Register Indirect	11111111 | 11 100 reg	5		7, 23
Memory Indirect	11111111 | mod 100 r/m	5	5	7
CALL = Call (within segment)				
Direct	11101000 | full displacement	3		7, 23
Register Indirect	11111111 | 11 010 reg	5		7, 23
Memory Indirect	11111111 | mod 010 r/m	5	5	7
RET = Return from CALL (within segment)				
	11000011	5	5	
Adding Immediate to SP	11000010 | 16-bit disp.	5	5	

Mnemonic cccc	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less Than or Equal/Greater Than	1111

Table 10.1. Intel486™ SX Microprocessor/Intel487™ SX Math CoProcessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
CONTROL TRANSFER (within segment) (Continued)				
ENTER = Enter Procedure	11001000 | 16-bit disp., 8-bit level			
Level = 0		14		
Level = 1		17		
Level (L) > 1		17+3L		8
LEAVE = Leave Procedure	11001001	5	1	
MULTIPLE-SEGMENT INSTRUCTIONS				
MOV = Move				
reg. to segment reg.	10001110 | 11 sreg3 reg	3/9	0/3	RV/P, 9
memory to segment reg.	10001110 | mod sreg3 r/m	3/9	2/5	RV/P, 9
segment reg. to reg.	10001100 | 11 sreg3 reg	3		
segment reg. to memory	10001100 | mod sreg3 r/m	3		
PUSH = Push				
segment reg. (ES, CS, SS, or DS)	000sreg2110	3		
segment reg. (FS or GS)	00001111 | 10 sreg3000	3		
POP = Pop				
segment reg. (ES, SS, or DS)	000sreg2111	3/9	2/5	RV/P, 9
segment reg. (FS or GS)	00001111 | 10 sreg3001	3/9	2/5	RV/P, 9
LDS = Load Pointer to DS	11000101 | mod reg r/m	6/12	7/10	RV/P, 9
LES = Load Pointer to ES	11000100 | mod reg r/m	6/12	7/10	RV/P, 9
LFS = Load Pointer to FS	00001111 | 10110100 | mod reg r/m	6/12	7/10	RV/P, 9
LGS = Load Pointer to GS	00001111 | 10110101 | mod reg r/m	6/12	7/10	RV/P, 9
LSS = Load Pointer to SS	00001111 | 10110010 | mod reg r/m	6/12	7/10	RV/P, 9
CALL = Call				
Direct intersegment	10011010 | unsigned full offset, selector	18	2	R, 7, 22
to same level		20	3	P, 9
thru Gate to same level		35	6	P, 9
to inner level, no parameters		69	17	P, 9
to inner level, x parameter (d) words		77+4X	17+n	P, 11, 9
to TSS		37+TS	3	P, 10, 9
thru Task Gate		38+TS	3	P, 10, 9
Indirect intersegment	11111111 | mod 011 r/m	17	8	R, 7
to same level		20	10	P, 9
thru Gate to same level		35	13	P, 9
to inner level, no parameters		69	24	P, 9
to inner level, x parameter (d) words		77+4X	24+n	P, 11, 9
to TSS		37+TS	10	P, 10, 9
thru Task Gate		38+TS	10	P, 10, 9
RET = Return from CALL				
intersegment	11001011	13	8	R, 7
to same level		17	9	P, 9
to outer level		35	12	P, 9
intersegment adding imm. to SP	11001010 | 16-bit disp.	14	8	R, 7
to same level		18	9	P, 9
to outer level		36	12	P, 9

Table 10.1. Intel486™ SX Microprocessor/Intel487™ SX Math CoProcessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
MULTIPLE-SEGMENT INSTRUCTIONS (Continued)				
JMP = Unconditional Jump				
Direct intersegment	11101010 | unsigned full offset, selector	17	2	R, 7, 22
to same level		19	3	P, 9
thru Call Gate to same level		32	6	P, 9
thru TSS		42+TS	3	P, 10, 9
thru Task Gate		43+TS	3	P, 10, 9
Indirect intersegment	11111111 | mod 101 r/m	13	9	R, 7, 9
to same level		18	10	P, 9
thru Call Gate to same level		31	13	P, 9
thru TSS		41+TS	10	P, 10, 9
thru Task Gate		42+TS	10	P, 10, 9
BIT MANIPULATION				
BT = Test bit				
register, immediate	00001111 | 10111010 | 11 100 reg | imm. 8-bit data	3		
memory, immediate	00001111 | 10111010 | mod 100 r/m | imm. 8-bit data	3	1	
reg1, reg2	00001111 | 10100011 | 11 reg2 reg1	3		
memory, reg	00001111 | 10100011 | mod reg r/m	8	2	
Instruction / **TTT**: BTS = Test Bit and Set: 101; BTR = Test Bit and Reset: 110; BTC = Test Bit and Compliment: 111				
register, immediate	00001111 | 10111010 | 11 TTT reg | imm. 8-bit data	6		
memory, immediate	00001111 | 10111010 | mod TTT r/m | imm. 8-bit data	8	2/0	U/L
reg1, reg2	00001111 | 10TTT011 | 11 reg2 reg1	6		
memory, reg	00001111 | 10TTT011 | mod reg r/m	13	3/1	U/L
BSF = Scan Bit Forward				
reg1, reg2	00001111 | 10111100 | 11 reg2 reg1	6/42		MN/MX, 12
memory, reg	00001111 | 10111100 | mod reg r/m	7/43	2	MN/MX, 13
BSR = Scan Bit Reverse				
reg1, reg2	00001111 | 10111101 | 11 reg2 reg1	6/103		MN/MX, 14
memory, reg	00001111 | 10111101 | mod reg r/m	7/104	1	MN/MX, 15
STRING INSTRUCTIONS				
CMPS = Compare Byte Word	1010011w	8	6	16
LODS = Load Byte/Word to AL/AX/EAX	1010110w	5	2	
MOVS = Move Byte/Word	1010010w	7	2	16
SCAS = Scan Byte/Word	1010111w	6	2	
STOS = Store Byte/Word from AL/AX/EX	1010101w	5		
XLAT = Translate String	11010111	4	2	

Table 10.1. Intel486™ SX Microprocessor/Intel487™ SX Math CoProcessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
REPEATED STRING INSTRUCTIONS				
Repeated by Count in CX or ECX (C = Count in CX or ECX)				
REPE CMPS = Compare String (Find Non-Match)	11110011 1010011w			
C = 0		5		
C > 0		7+7c		16, 17
REPNE CMPS = Compare String (Find Match)	11110010 1010011w			
C = 0		5		
C > 0		7+7c		16, 17
REP LODS = Load String	11110010 1010110w			
C = 0		5		
C > 0		7+4c		16, 18
REP MOVS = Move String	11110010 1010010w			
C = 0		5		
C = 1		13	1	16
C > 1		12+3c		16, 19
REPE SCAS = Scan String (Find Non-AL/AX/EAX)	11110011 1010111w			
C = 0		5		
C > 0		7+5c		20
REPNE SCAS = Scan String (Find AL/AX/EAX)	11110010 1010111w			
C = 0		5		
C > 0		7+5c		20
REP STOS = Store String	11110010 1010101w			
C = 0		5		
C > 0		7+4c		
FLAG CONTROL				
CLC = Clear Carry Flag	11111000	2		
STC = Set Carry Flag	11111001	2		
CMC = Complement Carry Flag	11110101	2		
CLD = Clear Direction Flag	11111100	2		
STD = Set Direction Flag	11111101	2		
CLI = Clear Interrupt Enable Flag	11111010	5		
STI = Set Interrupt Enable Flag	11111011	5		
LAHF = Load AH into Flag	10011111	3		
SAHF = Store AH into Flags	10011110	2		
PUSHF = Push Flags	10011100	4/3		RV/P
POPF = Pop Flags	10011101	9/6		RV/P
DECIMAL ARITHMETIC				
AAA = ASCII Adjust for Add	00110111	3		
AAS = ASCII Adjust for Subtract	00111111	3		
AAM = ASCII Adjust for Multiply	11010100 00001010	15		

Table 10.1. Intel486™ SX Microprocessor/Intel487™ SX Math CoProcessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
DECIMAL ARITHMETIC (Continued)				
AAD = ASCII Adjust for Divide	11010101 00001010	14		
DAA = Decimal Adjust for Add	00100111	2		
DAS = Decimal Adjust for Subtract	00101111	2		
PROCESSOR CONTROL INSTRUCTIONS				
HLT = Halt	11110100	4		
MOV = Move To and From Control/Debug/Test Registers				
CR0 from register	00001111 00100010 11 000 reg	17	2	
CR2/CR3 from register	00001111 00100010 11 eee reg	4		
Reg from CR0–3	00001111 00100000 11 eee reg	4		
DR0–3 from register	00001111 00100011 11 eee reg	10		
DR6–7 from register	00001111 00100011 11 eee reg	10		
Register from DR6–7	00001111 00100001 11 eee reg	9		
Register from DR0–3	00001111 00100001 11 eee reg	9		
TR3 from register	00001111 00100110 11 011 reg	4		
TR4–7 from register	00001111 00100110 11 eee reg	4		
Register from TR3	00001111 00100100 11 011 reg	3		
Register from TR4–7	00001111 00100100 11 eee reg	4		
CLTS = Clear Task Switched Flag	00001111 00000110	7	2	
INVD = Invalidate Data Cache	00001111 00001000	4		
WBINVD = Write-Back and Invalidate Data Cache	00001111 00001001	5		
INVLPG = Invalidate TLB Entry				
INVLPG memory	00001111 00000001 mod 111 r/m	12/11		H/NH
PREFIX BYTES				
Address Size Prefix	01100111	1		
LOCK = Bus Lock Prefix	11110000	1		
Operand Size Prefix	01100110	1		
Segment Override Prefix				
CS:	00101110	1		
DS:	00111110	1		
ES:	00100110	1		
FS:	01100100	1		
GS:	01100101	1		
SS:	00110110	1		

Table 10.1. Intel486™ SX Microprocessor/Intel487™ SX Math CoProcessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
PROTECTION CONTROL				
ARPL = Adjust Requested Privilege Level				
From register	01100011 | 11 reg1 reg2	9		
From memory	01100011 | mod reg r/m	9		
LAR = Load Access Rights				
From register	00001111 | 00000010 | 11 reg1 reg2	11	3	
From memory	00001111 | 00000010 | mod reg r/m	11	5	
LGDT = Load Global Descriptor				
Table register	00001111 | 00000001 | mod 010 r/m	12	5	
LIDT = Load Interrupt Descriptor				
Table register	00001111 | 00000001 | mod 011 r/m	12	5	
LLDT = Load Local Descriptor				
Table register from reg.	00001111 | 00000000 | 11 010 reg	11	3	
Table register from mem.	00001111 | 00000000 | mod 010 r/m	11	6	
LMSW = Load Machine Status Word				
From register	00001111 | 00000001 | 11 110 reg	13		
From memory	00001111 | 00000001 | mod 110 r/m	13	1	
LSL = Load Segment Limit				
From register	00001111 | 00000011 | 11 reg1 reg2	10	3	
From memory	00001111 | 00000011 | mod reg r/m	10	6	
LTR = Load Task Register				
From Register	00001111 | 00000000 | 11 011 reg	20		
From Memory	00001111 | 00000000 | mod 011 r/m	20		
SGDT = Store Global Descriptor Table	00001111 | 00000001 | mod 000 r/m	10		
SIDT = Store Interrupt Descriptor Table	00001111 | 00000001 | mod 001 r/m	10		
SLDT = Store Local Descriptor Table				
To register	00001111 | 00000000 | 11 000 reg	2		
To memory	00001111 | 00000000 | mod 000 r/m	3		
SMSW = Store Machine Status Word				
To register	00001111 | 00000001 | 11 100 reg	2		
To memory	00001111 | 00000001 | mod 100 r/m	3		
STR = Store Task Register				
To register	00001111 | 00000000 | 11 001 reg	2		
To memory	00001111 | 00000000 | mod 001 r/m	3		
VERR = Verify Read Access				
Register	00001111 | 00000000 | 11 100 r/m	11	3	
Memory	00001111 | 00000000 | mod 100 r/m	11	7	
VERW = Verify Write Access				
To register	00001111 | 00000000 | 11 101 reg	11	3	
To memory	00001111 | 00000000 | mod 101 r/m	11	7	

Table 10.1. Intel486™ SX Microprocessor/Intel487™ SX Math CoProcessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTERRUPT INSTRUCTIONS				
INT n = Interrupt Type n	11001101 | type	INT+4/0		RV/P, 21
INT 3 = Interrupt Type 3	11001100	INT+0		21
INTO = Interrupt 4 if Overflow Flag Set	11001110			
Taken		INT+2		21
Not Taken		3		21
BOUND = Interrupt 5 if Detect Value Out Range	01100010 | mod reg r/m			
If in range		7	7	21
If out of range		INT+24	7	21
IRET = Interrupt Return	11001111			
Real Mode/Virtual Mode		15	8	
Protected Mode				
To same level		20	11	9
To outer level		36	19	9
To nested task (EFLAGS.NT = 1)		TS+32	4	9, 10
External Interrupt		INT+11		21
NMI = Non-Maskable Interrupt		INT+3		21
Page Fault		INT+24		21
VM86 Exceptions				
CLI		INT+8		21
STI		INT+8		21
INT n		INT+9		
PUSHF		INT+9		21
POPF		INT+8		21
IRET		INT+9		
IN				
Fixed Port		INT+50		21
Variable Port		INT+51		21
OUT				
Fixed Port		INT+50		21
Variable Port		INT+51		21
INS		INT+50		21
OUTS		INT+50		21
REP INS		INT+51		21
REP OUTS		INT+51		21

Table 10.1(a). Task Switch Clock Counts Table

Method	Value for TS	
	Cache Hit	Miss Penalty
VM/Intel486™ DX CPU–Intel486 SX CPU–Intel487™ SX MCP/286 TSS To Intel486 DX CPU–Intel486 SX CPU–Intel487 SX MCP TSS	162	55
VM/Intel486 DX CPU–Intel486 SX CPU–Intel487 SX MCP/286 TSS To 286 TSS	143	31
VM/Intel486 DX CPU–Intel486 SX CPU–Intel487 SX MCP/286 TSS To VM TSS	140	37

Table 10.1(b). Interrupt Clock Counts Table

Method	Value for INT		
	Cache Hit	Miss Penalty	Notes
Real Mode	26	2	
Protected Mode			
Interrupt/Trap gate, same level	44	6	9
Interrupt/Trap gate, different level	71	17	9
Task Gate	37 + TS	3	9, 10
Virtual Mode			
Interrupt/Trap gate, different level	82	17	
Task gate	37 + TS	3	10

Abbreviations	Definition
16/32	16/32 bit modes
U/L	unlocked/locked
MN/MX	minimum/maximum
L/NL	loop/no loop
RV/P	real and virtual mode/protected mode
R	real mode
P	protected mode
T/NT	taken/not taken
H/NH	hit/no hit

NOTES:
1. Assuming that the operand address and stack address fall in different cache sets.
2. Always locked, no cache hit case.
3. Clocks = 10 + max($\log_2(|m|)$,n)
   m = multiplier value (min clocks for m=0)
   n = 3/5 for ±m
4. Clocks = {quotient(count/operand length)}*7+9
   = 8 if count ≤ operand length (8/16/32)
5. Clocks = {quotient(count/operand length)}*7+9
   = 9 if count ≤ operand length (8/16/32)
6. Equal/not equal cases (penalty is the same regardless of lock).
7. Assuming that addresses for memory read (for indirection), stack push/pop, and branch fall in different cache sets.
8. Penalty for cache miss: add 6 clocks for every 16 bytes copied to new stack frame.
9. Add 11 clocks for each unaccessed descriptor load.
10. Refer to task switch clock counts table for value of TS.
11. Add 4 extra clocks to the cache miss penalty for each 16 bytes.

For notes 12–13: (b = 0–3, non-zero byte number);
(i = 0–1, non-zero nibble number);
(n = 0–3, non bit number in nibble);

12. Clocks = 8+4 (b+1) + 3(i+1) + 3(n+1)
    = 6 if second operand = 0
13. Clocks = 9+4(b+1) + 3(i+1) + 3(n+1)
    = 7 if second operand = 0

For notes 14–15: (n = bit position 0–31)

14. Clocks = 7 + 3(32−n)
    6 if second operand = 0
15. Clocks = 8 + 3(32−n)
    7 if second operand = 0
16. Assuming that the two string addresses fall in different cache sets.
17. Cache miss penalty: add 6 clocks for every 16 bytes compared. Entire penalty on first compare.
18. Cache miss penalty: add 2 clocks for every 16 bytes of data. Entire penalty on first load.
19. Cache miss penalty: add 4 clocks for every 16 bytes moved.
    (1 clock for the first operation and 3 for the second)
20. Cache miss penalty: add 4 clocks for every 16 bytes scanned.
    (2 clocks each for first and second operations)
21. Refer to interrupt clock counts table for value of INT
22. Clock count includes one clock for using both displacement and immediate.
23. Refer to assumption 6 in the case of a cache miss.

Table 10.2. Intel486™ SX Microprocessor/Intel487™ SX Math CoProcessor I/O Instructions Clock Count Summary

INSTRUCTION	FORMAT	Real Mode	Protected Mode (CPL≤IOPL)	Protected Mode (CPL>IOPL)	Virtual 86 Mode	Notes
I/O INSTRUCTIONS						
IN = Input from:						
Fixed Port	1110010w | port number	14	9	29	27	
Variable Port	1110110w	14	8	28	27	
OUT = Output to:						
Fixed Port	1110011w | port number	16	11	31	29	
Variable Port	1110111w	16	10	30	29	
INS = Input Byte/Word from DX Port	0110110w	17	10	32	30	
OUTS = Output Byte/Word to DX Port	0110111w	17	10	32	30	1
REP INS = Input String	11110010 | 0110110w	16+8c	10+8c	30+8c	29+8c	2
REP OUTS = Output String	11110010 | 0110111w	17+5c	11+5c	31+5c	30+5c	3

NOTES:
1. Two clock cache miss penalty in all cases.
2. c = count in CX or ECX.
3. Cache miss penalty in all modes: Add 2 clocks for every 16 bytes. Entire penalty on second operation.

2

Table 10.3. Intel487™ SX Math CoProcessor Floating Point Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range ... Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range ... Upper Range)	Notes
DATA TRANSFER					
FLD = Real Load to ST(0)					
32-bit memory	11011 001 | mod 000 r/m | s-i-b/disp.	3	2		
64-bit memory	11011 101 | mod 000 r/m | s-i-b/disp.	3	3		
80-bit memory	11011 011 | mod 101 r/m | s-i-b/disp.	6	4		
ST(i)	11011 001 | 11000 ST(i)	4			
FILD = Integer Load to ST(0)					
16-bit memory	11011 111 | mod 000 r/m | s-i-b/disp.	14.5(13–16)	2	4	
32-bit memory	11011 011 | mod 000 r/m | s-i-b/disp.	11.5(9–12)	2	4(2–4)	
64-bit memory	11011 111 | mod 101 r/m | s-i-b/disp.	16.8(10–18)	3	7.8(2–8)	
FBLD = BCD Load to ST(0)	11011 111 | mod 100 r/m | s-i-b/disp.	75(70–103)	4	7.7(2–8)	
FST = Store Real from ST(0)					
32-bit memory	11011 001 | mod 010 r/m | s-i-b/disp.	7			1
64-bit memory	11011 101 | mod 010 r/m | s-i-b/disp.	8			2
ST(i)	11011 101 | 11010 ST(i)	3			
FSTP = Store Real from ST(0) and Pop					
32-bit memory	11011 011 | mod 011 r/m | s-i-b/disp.	7			1
64-bit memory	11011 101 | mod 011 r/m | s-i-b/disp.	8			2
80-bit memory	11011 011 | mod 111 r/m | s-i-b/disp.	6			
ST(i)	11011 101 | 11001 ST(i)	3			
FIST = Store Integer from ST(0)					
16-bit memory	11011 111 | mod 010 r/m | s-i-b/disp.	33.4(29–34)			
32-bit memory	11011 011 | mod 010 r/m | s-i-b/disp.	32.4(28–34)			
FISTP = Store Integer from ST(0) and Pop					
16-bit memory	11011 111 | mod 011 r/m | s-i-b/disp.	33.4(29–34)			
32-bit memory	11011 011 | mod 011 r/m | s-i-b/disp.	33.4(29–34)			
64-bit memory	11011 111 | mod 111 r/m | s-i-b/disp.	33.4(29–34)			
FBSTP = Store BCD from ST(0) and Pop	11011 111 | mod 110 r/m | s-i-b/disp.	175(172–176)			
FXCH = Exchange ST(0) and ST(i)	11011 001 | 11001 ST(i)	4			
COMPARISON INSTRUCTIONS					
FCOM = Compare ST(0) with Real					
32-bit memory	11011 000 | mod 010 r/m | s-i-b/disp.	4	2	1	
64-bit memory	11011 100 | mod 010 r/m | s-i-b/disp.	4	3	1	
ST(i)	11011 000 | 11010 ST(i)	4		1	
FCOMP = Compare ST(0) with Real and Pop					
32-bit memory	11011 000 | mod 011 r/m | s-i-b/disp.	4	2	1	
64-bit memory	11011 100 | mod 011 r/m | s-i-b/disp.	4	3	1	
ST(i)	11011 000 | 11011 ST(i)	4		1	

Table 10.3. Intel487™ SX Math CoProcessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range ... Upper Range)	Penalty If Cache Miss	Concurrent Execution Avg (Lower Range ... Upper Range)	Notes
COMPARISON INSTRUCTIONS (Continued)					
FCOMPP = Compare ST(0) with ST(1) and Pop Twice	11011 110 | 1101 1001	5		1	
FICOM = Compare ST(0) with Integer					
16-bit memory	11011 110 | mod 010 r/m | s-i-b/disp.	18(16–20)	2	1	
32-bit memory	11011 010 | mod 010 r/m | s-i-b/disp.	16.5(15–17)	2	1	
FICOMP = Compare ST(0) with Integer					
16-bit memory	11011 110 | mod 011 r/m | s-i-b/disp.	18(16–20)	2	1	
32-bit memory	11011 010 | mod 011 r/m | s-i-b/disp.	16.5(15–17)	2	1	
FTST = Compare ST(0) with 0.0	11011 001 | 1110 0100	4		1	
FUCOM = Unordered compare ST(0) with ST(i)	11011 101 | 11100 ST(i)	4		1	
FUCOMP = Unordered compare ST(0) with ST(i) and Pop	11011 101 | 11101 ST(i)	4		1	
FUCOMPP = Unordered compare ST(0) with ST(i) and Pop Twice	11011 101 | 11101 1001	5		1	
FXAM = Examine ST(0)	11011 001 | 1110 0101	8			
CONSTANTS					
FLDZ = Load +0.0 into ST(0)	11011 001 | 1110 1110	4			
FLD1 = Load +1.0 into ST(0)	11011 001 | 1110 1000	4			
FLDPI = Load π into ST(0)	11011 001 | 1110 1011	8		2	
FLDL2T = Load $\log_2(10)$ into ST(0)	11011 001 | 1110 1001	8		2	
FLDL2E = Load $\log_2(e)$ into ST(0)	11011 001 | 1110 1010	8		2	
FLDLG2 = Load $\log_{10}(2)$ into ST(0)	11011 001 | 1110 1100	8		2	
FLDLN2 = Load $\log_e(2)$ into ST(0)	11011 001 | 1110 1101	8		2	
ARITHMETIC					
FADD = Add Real with ST(0)					
ST(0) ← ST(0) + 32-bit memory	11011 000 | mod 000 r/m | s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0) ← ST(0) + 64-bit memory	11011 100 | mod 000 r/m | s-i-b/disp.	10(8–20)	3	7(5–17)	
ST(d) ← ST(0) + ST(i)	11011 d00 | 11000 ST(i)	10(8–20)		7(5–17)	
FADDP = Add real with ST(0) and Pop (ST(i) ← ST(0) + ST(i))	11011 110 | 11000 ST(i)	10(8–20)		7(5–17)	
FSUB = Subtract real from ST(0)					
ST(0) ← ST(0) − 32-bit memory	11011 000 | mod 100 r/m | s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0) ← ST(0) − 64-bit memory	11011 100 | mod 100 r/m | s-i-b/disp.	10(8-20)	3	7(5–17)	
ST(d) ← ST(0) − ST(i)	11011 d00 | 11101 ST(i)	10(8–20)		7(5–17)	
FSUBP = Subtract real from ST(0) and Pop (ST(i) ← ST(0) − ST(i))	11011 110 | 11101 ST(i)	10(8–20)		7(5–17)	

2

Table 10.3. Intel487™ SX Math CoProcessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range . . . Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range . . . Upper Range)	Notes
ARITHMETIC (Continued)					
FSUBR = Subtract real reversed (Subtract ST(0) from real)					
ST(0) ← 32-bit memory − ST(0)	11011 000 | mod 101 r/m | s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0) ← 64-bit memory − ST(0)	11011 100 | mod 101 r/m | s-i-b/disp.	10(8–20)	3	7(5–17)	
ST(d) ← ST(i) − ST(0)	11011 d00 | 11100 ST(i)	10(8–20)		7(5–17)	
FSUBRP = Subtract real reversed and Pop (ST(i) ← ST(i) − ST(0))	11011 110 | 11100 ST(i)	10(8–20)		7(5–17)	
FMUL = Multiply real with ST(0)					
ST(0) ← ST(0) × 32-bit memory	11011 000 | mod 001 r/m | s-i-b/disp.	11	2	8	
ST(0) ← ST(0) × 64-bit memory	11011 100 | mod 001 r/m | s-i-b/disp.	14	3	11	
ST(d) ← ST(0) × ST(i)	11011 d00 | 11001 ST(i)	16		13	
FMULP = Multiply ST(0) with ST(i) and Pop (ST(i) ← ST(0) × ST(i))	11011 110 | 11001 ST(i)	16		13	
FDIV = Divide ST(0) by Real					
ST(0) ← ST(0)/32-bit memory	11011 000 | mod 110 r/m | s-i-b/disp.	73	2	70	3
ST(0) ← ST(0)/64-bit memory	11011 100 | mod 100 r/m | s-i-b/disp.	73	3	70	3
ST(d) ← ST(0)/ST(i)	11011 d00 | 11111 ST(i)	73		70	3
FDIVP = Divide ST(0) by ST(i) and Pop (ST(i) ← ST(0)/ST(i))	11011 110 | 11111 ST(i)	73		70	3
FDIVR = Divide real reversed (Real/ST(0))					
ST(0) ← 32-bit memory/ST(0)	11011 000 | mod 111 r/m | s-i-b/disp.	73	2	70	3
ST(0) ← 64-bit memory/ST(0)	11011 100 | mod 111 r/m | s-i-b/disp.	73	3	70	3
ST(d) ← ST(i)/ST(0)	11011 d00 | 11110 ST(i)	73		70	3
FDIVRP = Divide real reversed and Pop (ST(i) ← ST(i)/ST(0))	11011 110 | 11110 ST(i)	73		70	3
FIADD = Add Integer to ST(0)					
ST(0) ← ST(0) + 16-bit memory	11011 110 | mod 000 r/m | s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0) ← ST(0) + 32-bit memory	11011 010 | mod 000 r/m | s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FISUB = Subtract Integer from ST(0)					
ST(0) ← ST(0) − 16-bit memory	11011 110 | mod 100 r/m | s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0) ← ST(0) − 32-bit memory	11011 010 | mod 100 r/m | s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FISUBR = Integer Subtract Reversed					
ST(0) ← 16-bit memory − ST(0)	11011 110 | mod 101 r/m | s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0) ← 32-bit memory − ST(0)	11011 010 | mod 101 r/m | s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FIMUL = Multiply Integer with ST(0)					
ST(0) ← ST(0) × 16-bit memory	11011 110 | mod 001 r/m | s-i-b/disp.	25(23–27)	2	8	
ST(0) ← ST(0) × 32-bit memory	11011 010 | mod 001 r/m | s-i-b/disp.	23.5(22–24)	2	8	
FIDIV = Integer Divide					
ST(0) ← ST(0)/16-bit memory	11011 110 | mod 110 r/m | s-i-b/disp.	87(85–89)	2	70	3
ST(0) ← ST(0)/32-bit memory	11011 010 | mod 110 r/m | s-i-b/disp.	85.5(84–86)	2	70	3

Table 10.3. Intel487™ SX Math CoProcessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range . . . Upper Range)	Penalty If Cache Miss	Concurrent Execution Avg (Lower Range . . . Upper Range)	Notes
ARITHMETIC (Continued)					
FIDIVR = Integer Divide Reversed					
ST(0) ← 16-bit memory/ST(0)	11011 110 | mod 111 r/m | s-i-b/disp.	87(85–89)	2	70	3
ST(0) ← 32-bit memory/ST(0)	11011 010 | mod 111 r/m | s-i-b/disp.	85.5(84–86)	2	70	3
FSQRT = Square Root	11011 001 | 1111 1010	85.5(83–87)		70	
FSCALE = Scale ST(0) by ST(1)	11011 001 | 1111 1101	31(30–32)		2	
FXTRACT = Extract components of ST(0)	11011 001 | 1111 0100	19(16–20)		4(2–4)	
FPREM = Partial Reminder	11011 001 | 1111 1000	84(70–138)		2(2–8)	
FPREM1 = Partial Reminder (IEEE)	11011 001 | 1111 0101	94.5(72–167)		5.5(2–18)	
FRNDINT = Round ST(0) to integer	11011 001 | 1111 1100	29.1(21–30)		7.4(2–8)	
FABS = Absolute value of ST(0)	11011 001 | 1110 0001	3			
FCHS = Change sign of ST(0)	11011 001 | 1110 0000	6			
TRANSCENDENTAL					
FCOS = Cosine of ST(0)	11011 001 | 1111 1111	241(193–279)		2	6, 7
FPTAN = Partial tangent of ST(0)	11011 001 | 1111 0010	244(200–273)		70	6, 7
FPATAN = Partial arctangent	11011 001 | 1111 0011	289(218–303)		5(2–17)	6
FSIN = Sine of ST(0)	11011 001 | 1111 1110	241(193–279)		2	6, 7
FSINCOS = Sine and cosine of ST(0)	11011 001 | 1111 1011	291(243–329)		2	6, 7
F2XM1 = $2^{ST(0)} - 1$	11011 001 | 1111 0000	242(140–279)		2	6
FYL2X = ST(1) × $\log_2$(ST(0))	11011 001 | 1111 0001	311(196–329)		13	6
FYL2XP1 = ST(1) × $\log_2$(ST(0) + 1.0)	11011 001 | 1111 1001	313(171–326)		13	6
PROCESSOR CONTROL					
FINIT = Initialize FPU	11011 011 | 1110 0011	17			4
FSTSW AX = Store status word into AX	11011 111 | 1110 0000	3			5
FSTSW = Store status word into memory	11011 101 | mod 111 r/m | s-i-b/disp.	3			5
FLDCW = Load control word	11011 001 | mod 101 r/m | s-i-b/disp.	4	2		
FSTCW = Store control word	11011 001 | mod 111 r/m | s-i-b/disp.	3			5
FCLEX = Clear exceptions	11011 011 | 1110 0010	7			4
FSTENV = Store environment	11011 001 | mod 110 r/m | s-i-b/disp.				
Real and Virtual modes 16-bit Address		67			4
Real and Virtual modes 32-bit Address		67			4
Protected mode 16-bit Address		56			4
Protected mode 32-bit Address		56			4
FLDENV = Load environment	11011 001 | mod 100 r/m | s-i-b/disp.				
Real and Virtual modes 16-bit Address		44	2		
Real and Virtual modes 32-bit Address		44	2		
Protected mode 16-bit Address		34	2		
Protected mode 32-bit Address		34	2		

2

Table 10.3. Intel487™ SX Math CoProcessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range ... Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range ... Upper Range)	Notes
PROCESSOR CONTROL (Continued)					
FSAVE = Save state	11011 101 | mod 110 r/m | s-i-b/disp.				
Real and Virtual modes 16-bit Address		154			4
Real and Virtual modes 32-bit Address		154			4
Protected mode 16-bit Address		143			4
Protected mode 32-bit Address		143			4
FRSTOR = Restore state	11011 101 | mod 100 r/m | s-i-b/				
Real and Virtual modes 16-bit Address		131	23		
Real and Virtual modes 32-bit Address		131	27		
Protected mode 16-bit Address		120	23		
Protected mode 32-bit Address		120	27		
FINCSTP = Increment Stack Pointer	11011 001 | 1111 0111	3			
FDECSTP = Decrement Stack Pointer	11011 001 | 1111 0110	3			
FFREE = Free ST(i)	11011 101 | 11000 ST(i)	3			
FNOP = No operations	11011 001 | 1101 0000	3			
WAIT = Wait until FPU ready (Minimum/Maximum)	10011011	1/3			

NOTES:
1. If operand is 0 clock counts = 27.
2. If operand is 0 clock counts = 28.
3. If CW.PC indicates 24 bit precision then subtract 38 clocks.
   If CW.PC indicates 53 bit precision then subtract 11 clocks.
4. If there is a numeric error pending from a previous instruction add 17 clocks.
5. If there is a numeric error pending from a previous instruction add 18 clocks.
6. The INT pin is polled several times while this instruction is executing to assure short interrupt latency.
7. If ABS(operand) is greater than $\pi/4$ then add n clocks. Where n = (operand/($\pi/4$)).

10.2 Instruction Encoding

10.2.1 OVERVIEW

All instruction encodings are subsets of the general instruction format shown in Figure 10.1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the "mod r/m" byte and "scaled index" byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16 or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 10.1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the Figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 10.4 is a complete list of all fields appearing in the Intel486 SX microprocessor/Intel OverDrive Processor instruction set. Further ahead, following Table 10.4, are detailed tables for each field.

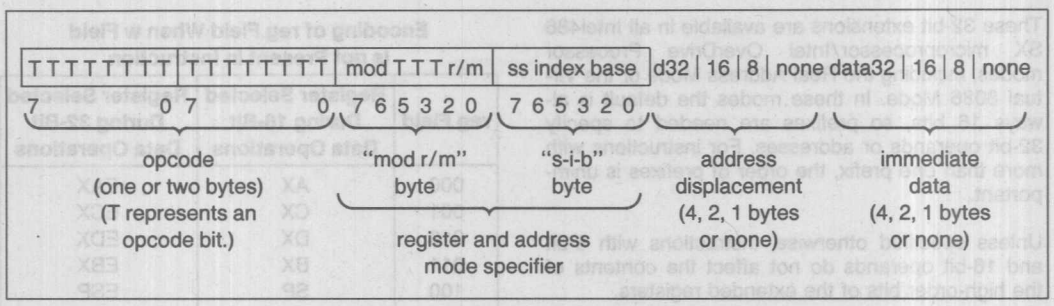


Figure 10.1. General Instruction Format

Table 10.4. Fields within Intel486™ SX Microprocessor/ Intel OverDrive Processor Instructions

Field Name	Description	Number of Bits
w	Specifies if Data is Byte or Full Size (Full Size is either 16 or 32 Bits	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field Must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
tttn	For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated	4

NOTE:
Tables 10.1–10.3 show encoding of individual instructions.

10.2.2 32-BIT EXTENSIONS OF THE INSTRUCTION SET

With the Intel486 SX microprocessor/Intel OverDrive Processor, the 8086/80186/80286 instruction set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

Whether the instruction defaults to operations of 16 bits or 32 bits depends on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 bits or 16 bits) for both operands and effective addresses when executing that code segment. In the Real Address Mode or Virtual 8086 Mode, no code segment descriptors are used, but a D value of 0 is assumed internally by the Intel486 SX microprocessor/Intel OverDrive Processor when operating in those modes (for 16-bit default sizes compatible with the 8086/80186/80286).

Two prefixes, the Operand Size Prefix and the Effective Address Size Prefix, allow overriding individually the Default selection of operand size and effective address size. These prefixes may precede any opcode bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed before the opcode bytes. The presence of the Operand Size Prefix and the Effective Address Prefix will toggle the operand size or the effective address size, respectively, to the value "opposite" from the Default setting. For example, if the default operand size is for 32-bit data operations, then presence of the Operand Size Prefix toggles the instruction to 16-bit data operation. As another example, if the default effective address size is 16 bits, presence of the Effective Address Size prefix toggles the instruction to use 32-bit effective address computations.

These 32-bit extensions are available in all Intel486 SX microprocessor/Intel OverDrive Processor modes, including the Real Address Mode or the Virtual 8086 Mode. In these modes the default is always 16 bits, so prefixes are needed to specify 32-bit operands or addresses. For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

10.2.3 ENCODING OF INTEGER INSTRUCTION FIELDS

Within the instruction are several fields indicating register selection, addressing mode and so on. The exact encodings of these fields are defined immediately ahead.

10.2.3.1 Encoding of Operand Length (w) Field

For any given instruction performing a data operation, the instruction is executing as a 32-bit operation or a 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size During 16-Bit Data Operations	Operand Size During 32-Bit Data Operations
0	8 Bits	8 Bits
1	16 Bits	32 Bits

10.2.3.2 Encoding of the General Register (reg) Field

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the "mod r/m" byte, or as the r/m field of the "mod r/m" byte.

Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected During 16-Bit Data Operations	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations:		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field During 32-Bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

10.2.3.3 Encoding of the Segment Register (sreg) Field

The sreg field in certain instructions is a 2-bit field allowing one of the four 80286 segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the Intel486 SX microprocessor/ Intel OverDrive Processor FS and GS segment registers to be specified.

2-Bit sreg2 Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

10.2.3.4 Encoding of Address Mode

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the "mod r/m" byte, and a second byte of addressing information, the "s-i-b" (scale-index-base) byte, can be specified.

The s-i-b byte (scale-index-base byte) is specified when using 32-bit addressing mode and the "mod r/m" byte has r/m = 100 and mod = 00, 01 or 10. When the sib byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the "mod r/m" byte, also contains three bits (shown as TTT in Figure 10.1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the "mod r/m" byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the "mod r/m" byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following three pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Encoding of 16-bit Address Mode with "mod r/m" Byte

mod r/m	Effective Address
00 000	DS:[BX + SI]
00 001	DS:[BX + DI]
00 010	SS:[BP + SI]
00 011	SS:[BP + DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BP + DI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by r/m During 16-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m During 32-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode with "mod r/m" byte (no "s-i-b" byte present)

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by reg or r/m During 16-Bit Data Operations:		
mod r/m	Function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by reg or r/m During 32-Bit Data Operations:		
mod r/m	Function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode ("mod r/m" byte and "s-i-b" byte present)

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

NOTE:
Mod field in "mod r/m" byte; ss, index, base fields in "s-i-b" byte.

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg**
101	EBP
110	ESI
111	EDI

****IMPORTANT NOTE:**
When index field is 100, indicating "no index register," then ss field MUST equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

10.2.3.5 Encoding of Operation Direction (d) Field

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory <- - Register "reg" Field Indicates Source Operand; "mod r/m" or "mod ss index base" Indicates Destination Operand
1	Register <- - Register/Memory "reg" Field Indicates Destination Operand; "mod r/m" or "mod ss index base" Indicates Source Operand

10.2.3.6 Encoding of Sign-Extend (s) Field

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

s	Effect on Immediate Data8	Effect on Immediate Data 16|32
0	None	None
1	Sign-Extend Data8 to Fill 16-Bit or 32-Bit Destination	None

10.2.3.7 Encoding of Conditional Test (tttn) Field

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with n indicating to use the condition (n = 0) or its negation (n = 1), and ttt giving the condition to test.

Mnemonic	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

10.2.3.8 Encoding of Control or Debug or Test Register (eee) Field

For the loading and storing of the Control, Debug and Test registers.

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	CR2
011	CR3
Do not use any other encoding	

When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding	

When Interpreted as Test Register Field

eee Code	Reg Name
011	TR3
100	TR4
101	TR5
110	TR6
111	TR7
Do not use any other encoding	

	Instruction								Optional Fields	
	First Byte				Second Byte					
1	11011	OPA		1	mod		1	OPB	r/m	s-i-b	disp
2	11011	MF		OPA	mod		OPB		r/m	s-i-b	disp
3	11011	d	P	OPA	1	1	OPB		ST(i)		
4	11011	0	0	1	1	1	1	OP			
5	11011	0	1	1	1	1	1	OP			
	15–11	10	9	8	7	6	5	4 3 2 1 0			

10.2.4 ENCODING OF FLOATING POINT INSTRUCTION FIELDS

Instructions for the FPU assume one of the five forms shown in the following table. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011B.

OP = Instruction opcode, possible split into two fields OPA and OPB

MF = Memory Format
- 00—32-bit real
- 01—32-bit integer
- 10—64-bit real
- 11—16-bit integer

P = Pop
- 0—Do not pop stack
- 1—Pop stack after operation

d = Destination
- 0—Destination is ST(0)
- 1—Destination is ST(i)

R XOR d = 0—Destination (op) Source
R XOR d = 1—Source (op) Destination

ST(i) = Register stack element *i*
- 000 = Stack top
- 001 = Second stack element
- •
- •
- •
- 111 = Eighth stack element

mod (Mode field) and r/m (Register/Memory specifier) have the same interpretation as the corresponding fields of the integer instructions.

s-i-b (Scale Index Base) byte and disp (displacement) are optionally present in instructions that have mod and r/m fields. Their presence depends on the values of mod and r/m, as for integer instructions.

11.0 DIFFERENCES BETWEEN THE Intel486™ SX MICROPROCESSOR/Intel OverDrive PROCESSOR AND THE Intel386™ MICROPROCESSOR PLUS THE Intel387™ MATH COPROCESSOR EXTENSION

The differences between the Intel486 SX microprocessor/Intel OverDrive Processor and the Intel386 microprocessor are due to performance enhancements. The differences are listed below.

1. Instruction clock counts have been reduced to achieve higher performance. See Section 10.
2. The Intel486 SX microprocessor/Intel OverDrive Processor bus is significantly faster than the Intel386 microprocessor bus. Differences include a 1X clock, parity support, burst cycles, cacheable cycles, cache invalidate cycles and 8-bit bus support. The Hardware Interface and Bus Operation Sections (Sections 6 and 7) of the data sheet should be carefully read to understand the Intel486 SX microprocessor/Intel OverDrive Processor bus functionality.
3. To support the on-chip cache new bits have been added to control register 0 (CD and NW) (Section 2.1.2.1), new pins have been added to the bus (Section 6) and new bus cycle types have been added (Section 7). The on-chip cache needs to be enabled after reset by clearing the CD and NW bit in CR0.
4. Six new instructions have been added:

   Byte Swap (BSWAP)

   Exchange-and-Add (XADD)

   Compare and Exchange (CMPXCHG)

   Invalidate Data Cache (INVD)

   Write-back and Invalidate Data Cache (WBINVD)

   Invalidate TLB Entry (INVLPG)
5. There are two new bits defined in control register 3, the page table entries and page directory entries (PCD and PWT) (Section 4.5.2.5).
6. A new page protection feature has been added. This feature required a new bit in control register 0 (WP) (Section 2.1.2.1 and 4.5.3).
7. A new Alignment Check feature has been added. This feature required a new bit in the flags register (AC) (Section 2.1.1.3) and a new bit in control register 0 (AM) (Section 2.1.2.1).
8. The replacement algorithm for the translation lookaside buffer has been changed from a random algorithm to a pseudo least recently used algorithm like that used by the on-chip cache. See Section 5.5 for a description of the algorithm.
9. Three new testability registers, TR3, TR4 and TR5, have been added for testing the on-chip cache. TLB testability has been enhanced. See Section 8.
10. The prefetch queue has been increased from 16 bytes to 32 bytes. A jump always needs to execute after modifying code to guarantee correct execution of the new instruction.
11. After reset, the ID in the upper byte of the DX register is 04. The contents of the base registers including the floating point registers may be different after reset.

In addition to the previously mentioned enhancements, the Intel OverDrive Processor offers the following features:

1. The complete Intel387 Math CoProcessor instruction set and register set have been added. No I/O cycles are performed during Floating Point instructions. The instruction and data pointers are set to 0 after FINIT/FSAVE. Interrupt 9 can no longer occur, interrupt 13 occurs instead.
2. The Intel OverDrive Processor supports new floating point error reporting modes to guarantee DOS compatibility. These new modes required a new bit in control register 0 (NE) (Section 2.1.2.1) and new pins (FERR# and IGNNE#) (Section 6.2.13 and 7.2.14).
3. In some cases FERR# is asserted when the next floating point instruction is encountered and in other cases it is asserted before the next floating point instruction is encountered, depending upon the execution state the instruction causing exception (see Sections 6.2.13 and 7.2.14). For both of these cases, the Intel387 Math CoProcessor asserts ERROR# when the error occurs and does not wait for the next floating point instruction to be encountered.
4. After reset, the ID in the upper byte of the DX register is 04. The contents of floating point registers may be different after reset.

12.0 DIFFERENCES WITH THE Intel486™ SX MICROPROCESSOR IN PGA vs PQFP

The differences between the Intel486 SX Microprocessor in PGA package versus PQFP package are listed below for quick reference when converting a system design from PGA to one in PQFP.

1. Sections 13.4.1 and 13.4.2 show the derating curves for the Intel486 SX Microprocessor in PGA and PQFP respectively. The Intel486 SX Microprocessor in PQFP has smaller output buffers than the ones in the PGA, hence the difference in the derating curves. The systems/boards must be evaluated for sufficient timing margins when converting from the Intel486 SX Microprocessor in PGA to the Intel486 SX Microprocessor in PQFP. Please note that both versions of the CPU have the same AC characteristics at 50 pF loads. For other loads (pF), an appropriate delay (ns) should be added (or subtracted to the AC timings depending on the package (PGA or PQFP) being used for the Intel486 SX Microprocessor.

2. D.C. Characteristics:

   Although both versions of the Intel486 SX Microprocessor have the same A.C. Characteristics, the D.C. Characteristics are different as listed below.

Parameter		PGA	PQFP
I_{CC} max (mA)	@16 MHz	525	450
	@20 MHz	600	500
	@25 MHz	700	560
	@33 MHz	780	575
C_{IN} (pF)	Input Capacitance	20	10
C_O (pF)	I/O Capacitance	20	10
C_{CLK} (pF)	CLK Capacitance	20	6

3. Power Down Mode:

   This mode is available in the Intel486 SX Microprocessor in PQFP only. See section 2.0.1 for detailed information. The OverDrive Processor is supported by the Intel486 SX Microprocessor in PQFP only. See section 2.0.1 and 6.6.2.1.

4. Boundary Scan (JTAG):

   The boundary scan feature is supported in the Intel486 SX Microprocessor in PQFP only. See section 8.5.

13.0 Intel OverDrive™ PROCESSOR

The Intel OverDrive Processor is essentially an enhanced Intel486 Microprocessor. There are three functional differences between the Intel OverDrive Processor and Intel486 Microprocessors. First, the Intel OverDrive Processor has an internal clock doubling circuit which decreases the time required to execute instructions. Second, the Intel OverDrive Processor does not support the JTAG boundary scan test feature (available with the PQFP version of the Intel486 SX Microprocessor). Third, the Intel OverDrive Processor has a different CPU revision identification than the Intel486 SX or Intel486 DX CPUs. These three differences are described in the following sections according to how they effect the CPU functionality. Additional OverDrive Processor specific information is provided in the "Intel OverDrive Processor for Intel486 SX Microprocessor-Based Systems," order number 290436-001.

13.1 Hardware Interface

The Intel OverDrive Processor bus has been designed to be identical with the Intel486 Microprocessor bus. Although the external clock is internally doubled and data and instructions are manipulated in the CPU core at twice the external frequency, the external bus is functionally identical with the Intel486 CPU.

The four boundary scan test signals (TCK, Test clock; TMS, Test Mode select; TDI, Test Data Input; TDO, Test Data Output), defined for the PQFP Intel 486 SX CPU, are not specified for the Intel OverDrive Processor.

The UP# (Upgrade Present) signal, which is defined as an input for the PQFP Intel486 SX CPU, is an output signal on the Intel OverDrive Processor. The UP# pin on the Intel OverDrive Processor provides a logical low output signal which can be used to enable logic to recognize and configure the system for the Intel OverDrive Processor.

The DX register always contains the component identifier at the conclusion of RESET. The Intel OverDrive Processor has a different revision identifier in the DL register than the Intel486 SX or Intel486 DX Microprocessors. When the OverDrive Processor is installed in a system the component identifier is supplied by the OverDrive Processor, rather than the original CPU. The stepping identification portion of the component identification will change with different revisions of the OverDrive Processor. The designer should only assume that the component identification for the OverDrive Processor will be 043xH, where 'x' is the stepping identifier.

13.2 Testability

As detailed in Section 13.1, the Intel OverDrive Processor does not support the JTAG boundary scan testability feature.

13.3 Instruction Set Summary

The Intel OverDrive Processor supports all Intel486 extensions to the 8086/80186/80286 instruction set. In general, instructions will execute faster on the Intel OverDrive Processor than the Intel486 Microprocessor. Specifically, an instruction that only uses memory from the on-chip cache executes at the full core clock rate while all bus accesses execute at the bus clock rate. To calculate the elapsed time of an instruction, the number of clock counts for that instruction must be multiplied by the clock period for the system. The instruction set clock count summary tables from Section 10.0 can be used for the OverDrive Processor with the following modifications:

— Clock counts for a cache hit: This value represents the number of internal CPU core clocks for an instruction that requires no external bus accesses or the base core clocks for an instruction requiring external bus accesses.

— Penalty clock counts for a cache miss: This value represents the worst-case approximation of the additional number of external clock counts that are required for an instruction which must access the external bus for data (a cache miss). This number must be multiplied by 2 to convert it to an equal number of internal CPU core clock counts and added to the base core clocks to compute the total number of core clocks for this instruction.

The actual number of core clocks for an instruction with a cache miss may be less than the base clock counts (from the cache hit column) plus the penalty clock counts (2 times the cache miss column number). The clock counts in the cache miss penalty column can be a cumulative value of external bus clocks (for data reads) and internal clocks for manipulating the data which has been loaded from the external bus. The number of clocks which are related to external bus accesses are correctly represented in terms of internal core clocks by multiplying by two. However, the clock counts related to internal data manipulation should not be multiplied by two. Therefore the total number of CPU core clock counts for an instruction with a cache miss represents a worst-case approximation.

To calculate the execution time for an OverDrive Processor instruction, multiply the total CPU core clock counts by the core clock period. For example, in a 25 MHz system the core clock period is 50 ns (1/50 MHz).

Additionally, the assumptions specified below should be understood in order to estimate instruction execution time.

A cache miss will force the OverDrive Processor to run an external bus cycle. The OverDrive Processor 32-bit burst bus is defined as r−b−w.

Where:

r = The number of bus clocks in the first cycle of a burst read or the number of clocks per data cycle is a non-burst read.

b = The number of bus clocks for the second and subsequent cycles in a burst read.

w = The number of bus clocks for a write.

The fastest bus the OverDrive Processor can support is 2−1−2 assuming 0 waits states. The clock counts in the cache miss penalty column assume a 2−1−2 bus. For slower busses add r−2 clocks to the cache miss penalty for the first dword accessed. Other factors also affect instruction clock counts.

Instruction Clock Count Assumptions

1. The external bus is available for reads or writes at all times. Else add bus clocks to reads until the bus is available
2. Accesses are aligned. Add three core clocks to each misaligned access.
3. Cache fills complete before subsequent accesses to the same line. If a read misses the cache during a cache fill due to a previous read or prefetch, the read must wait for the cache fill to complete. If a read or write accesses a cache line still being filled, it must wait for the fill to complete.
4. If an effective address is calculated, the base register is not the distination register of the preceding instruction. If the base register is the destination register of the preceding instruction add 1 to the core clock counts shown. Back-to-back PUSH and POP instructions are not affected by this rule.
5. An effective address calculation uses one base register and does not use an index register. However, if the effective address calculation uses an index register. 1 core clock may be added to the clock shown.
6. The target of a jump is in the cache. If not, add r clocks for accessing the destination instruction of a jump. If the destination instruction is not completely contained in the first dword read, add a maximum of 3b bus clocks. If the destination instruction is not completely contained in the first 16 byte burst, add a maximum of another r+3b bus clocks.
7. If no write buffer delay, w bus clocks are added only in the case in which all write buffers are full.
8. Displacement and immediate not used together. If displacement and immediate used together, 1 core clock may be added to the core clock count shown.
9. No invalidate cycles. Add a delay of 1 bus clock for each invalidate cycle if the invalidate cycle contends for the internal cache/external bus when the OverDrive Processor needs to use it.
10. Page translation hits in TLB. A TLB miss will add 13, 21 or 28 bus clocks + 1 possible core clock to the instruction depending on whether the Accessed and/or Dirty bit in neither, one or both of the page entries needs to be set in memory. This assumes that neither page entry is in the data cache and a page fault does not occur on the address translation.
11. No exceptions are detected during instruction execution. Refer to interrupt core Clock Counts Table for extra clocks if an interrupt is detected.
12. Instructions that read multiple consecutive data items (i.e., task switch, POPA, etc.) and miss the cache are assumed to start the first access on a 16-byte boundary. If not, an extra cache line fill may be necessary which may add up to (r+3b) bus clocks to the cache miss penalty.

13.4 BIOS and Software

The following should be considered when designing a system for upgrade with an Intel OverDrive Processor.

13.4.1 Intel OverDrive PROCESSOR DETECTION

The component identifier and stepping/revision identifier for the Intel OverDrive Processor is readable in DH and DL registers respectively, immediately after RESET, where

DH = 04h

DL = 30h–3Fh.

As it is difficult to differentiate between the Intel486 DX CPU and the Intel OverDrive Processor in software, it is recommended that the BIOS save the contents of the DX register, immediately after RESET, so that this information can be used later, if required, to identify an Intel OverDrive Processor in the system.

NOTE:
Initialization routines for Intel486 SX CPU systems should check for the presence of a floating point unit and set the CR0 register accordingly.

13.4.2 TIMING DEPENDENT LOOPS

The Intel OverDrive Processor executes instructions at twice the frequency of the input clock. Thus, software (or instruction based) timing loops will execute faster on the Intel OverDrive Processor than on the Intel486 DX or Intel486 SX CPU (at the same input clock frequency). Instructions such as NOP, LOOP, and JMP $+2, have been used by BIOS to implement timing loops that are required, for example, to enforce recovery time between consecutive accesses for I/O devices. These instruction based timing loop implementations may require modification for systems intended to be upgradable with the Intel OverDrive Processor.

In order to avoid any incompatibilities, it is recommended that timing requirements be implemented in hardware rather than in software. This provides transparency and also does not require any change in BIOS or I/O device drivers in the future when moving to higher processor clock speeds. As an example, a timing routine may be implemented as follows: The software performs a dummy I/O instruction to an unused I/O port. The hardware for the bus controller logic recognizes this I/O instruction and delays the termination of the I/O cycle to the CPU by keeping RDY# or BRDY# deasserted for the appropriate amount of time.

13.5 Thermal Management

The heat generated by the Intel OverDrive Processor requires that heat dissipation be managed carefully. The 25 MHz, 33 MHz Intel OverDrive Processor is supplied with a heat sink attached with adhesive to the package. 25 MHz, 33 MHz system designs must, therefore, provide space for the heat sink on the Intel OverDrive Processor.

13.5.1 THE Intel OverDrive™ PROCESSOR WITH ATTACHED HEAT SINK

The heat sink for the Intel OverDrive Processor is adhesively attached to the standard 169-lead, PGA package. Figure 13.1 shows a drawing of the Intel OverDrive Processor with the heat sink (see Table 13.1 for dimensions).

The maximum and minimum dimensions for the 169-pin, PGA package with heat sink are shown in Table 13.1.

Table 13.1. Intel OverDrive™ Processor, 169-Pin, PGA Package Dimensions with Heat Sink Attached

Dimension (Inches)	Min	Max
A. Heat Sink Width	1.520	1.550
B. PGA Package Width	1.735	1.765
C. Heat Sink Edge Gap	0.065	0.155
D. Heat Sink Height	0.212	0.260
E. Adhesive Thickness	0.008	0.012
F. Package Height from Stand-Offs	0.140	0.180
G. Total Height from Package Stand-Offs to Top of Heat Sink	0.360	0.452

The standard product markings and logo for the Intel OverDrive Processor with the attached heat sink will be included on a 1 in^2 plate located on the top, cen-

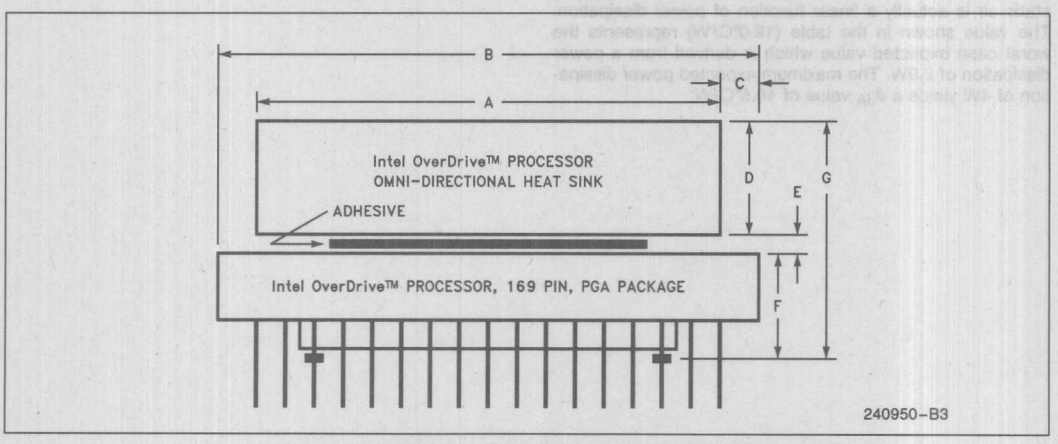


Figure 13.1. 169-Pin, PGA Package with Heat Sink

ter of the heat sink. The heat sink is omni-directional which allows the air to flow from any direction to achieve adequate cooling. The thermal resistance values for the Intel OverDrive Processor with attached heat sink are shown in Table 13.2.

Table 13.2. Thermal Resistance for the Intel OverDrive™ Processor with Attached Heat Sink

θ_{JS} = 2.5°C/W	Airflow (Ft/min, LFM)				
	0	200	400	600	800
θ_{JA}(°C/W)	14.0*	10.0	7.5	6.2	5.7

NOTE:
*The thermal resistance from junction to ambient (θ_{JA}) in static air is actually a linear function of power dissipation. The value shown in the table (14.0°C/W) represents the worst case expected value which is derived from a power dissipation of 2.9W. The maximum expected power dissipation of 6W yields a θ_{JA} value of 13.1°C/W.

13.5.2 Intel OverDrive™ PROCESSOR WITHOUT HEAT SINK

The 20 MHz Intel OverDrive Processor, for 16 MHz and 20 MHz i486 SX CPU systems, is supplied without a heat sink. Table 13.3 contains the thermal resistance values for the Intel OverDrive Processor without heat sink.

Table 13.3. Thermal Resistance for the Intel OverDrive™ Processor without Heat Sink

θ_{JC} = 2.0°C/W	Airflow (Ft/min, LFM)				
	0	200	400	600	800
θ_{JA}(°C/W)	19.0*	16.0	12.5	11.0	10.0

NOTE:
*The thermal resistance from junction to ambient (θ_{JA}) in static air is actually a linear function of power dissipation. The value shown in the table (19.0°C/W) represents the worst case expected value which is derived from a power dissipation of 2.9W. The maximum expected power dissipation of 4W yields a θ_{JA} value of 18.5°C/W.

13.5.3 THERMAL EQUATIONS

The methodology for calculating the heat dissipation for the 20 MHz Intel OverDrive Processor (without a heat sink) and the 25 MHz Intel OverDrive Processor (with a heat sink) are identical except that the reference point for measuring the product temperature is different. The 20 MHz Intel OverDrive Processor specifies T_{case} (the temperature at the outside center of the PGA package, opposite the pins) and θ_{JC} (the thermal resistance from the silicon junction to the package case) but the 25 MHz, 33 MHz Intel OverDrive Processor specifies T_{sink} (the temperature at the outside center base of the heat sink, not on the heat sink marking plate or cooling posts) and θ_{JS} (the thermal resistance from the silicon junction to the heat sink base). The relationships between temperature, thermal resistance and power are shown in the following equations:

$T_{case} = T_{ambient} + (P_{max} * \theta_{CA})$ and

$T_{sink} = T_{ambient} + (P_{max} * \theta_{SA})$

where,

$T_{ambient}$ = Ambient Temperature

P_{max} = Power ($I_{CC} * V_{CC}$),

$\theta_{CA} = \theta_{JA} - \theta_{JC}$,

$\theta_{SA} = \theta_{JA} - \theta_{JS}$.

14.0 ELECTRICAL DATA

The following sections describe recommended electrical connections for the Intel486 SX microprocessor/Intel OverDrive Processor, and its electrical specifications. Because all Intel486 SX microprocessor designs should accommodate either the OverDrive Processor or the Intel487 SX Math CoProcessor, the specifications given for the OverDrive Processor are the worst case values from the Intel487 SX Math CoProcessor and OverDrive Processor.

14.1 Power and Grounding

14.1.1 POWER CONNECTIONS

The Intel486 SX microprocessor/Intel487 SX Math CoProcessor is implemented in CHMOS IV technology and has modest power requirements. The Intel OverDrive Processor is implemented in CHMOS V technology. However, the high clock frequency output buffers can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 24 V_{CC} and 28 V_{SS} pins feed the Intel486 SX microprocessor/Intel OverDrive Processor.

Power and ground connections must be made to all external V_{CC} and GND pins of the Intel486 SX microprocessor/Intel OverDrive Processor. On the circuit board, all V_{CC} pins must be connected on a V_{CC} plane. All V_{SS} pins must be likewise connected on a GND plane.

14.1.2 POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitance should be placed near the Intel486 SX microprocessor/Intel OverDrive Processor. The Intel486 SX microprocessor/Intel OverDrive Processor driving its 32-bit parallel address and data busses at high frequencies can cause transient power surges, particularly when driving large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the Intel486 SX microprocessor/Intel OverDrive Processor and decoupling capacitors as much as possible. Capacitors specifically for PGA packages are also commercially available.

14.1.3 OTHER CONNECTION RECOMMENDATIONS

N.C. pins should always remain unconnected.

For reliable operation, always connect unused inputs to an appropriate signal level. Active LOW inputs should be connected to V_{CC} through a pullup resistor. Pullups in the range of 20 KΩ are recommended. Active HIGH inputs should be connected to GND.

14.2 Maximum Ratings

Table 14.1 is a stress rating only, and functional operation at the maximums is not guaranteed. Function operating conditions are given in 14.3 D.C. Specifications and 14.4 A.C. Specifications.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the Intel486 SX microprocessor/Intel OverDrive Processor contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

Table 14.1. Absolute Maximum Ratings

Case Temperature under Bias . . . −65°C to +110°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin with Respect to Ground . . . . . −0.5 to V_{CC} + 0.5V
Supply Voltage with Respect to V_{SS} . . . . . −0.5V to +6.5V

14.3 D.C. Specifications

Functional Operating Range: V_{CC} = 5V ±5%; (Note 1)

Table 14.2. Intel486™ SX Microprocessor D.C. Parametric Values (for PGA package)

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 2)
V_{OH}	Output High Voltage	2.4		V	(Note 3)
I_{CC}	Power Supply Current (16 MHz) Power Supply Current (20 MHz) Power Supply Current (25 MHz) Power Supply Current (33 MHz)		450 530 630 685	mA	(Note 4)
I_{CCF}	Power Supply Current with Intel486 SX Microprocessor Tri-Stated (Floating) CLK = 16 MHz CLK = 20 MHz CLK = 25 MHz CLK = 33 MHz		 395 430 530 545	mA	(Note 4)
I_{LI}	Input Leakage Current		±15	μA	(Note 5)
I_{IH}	Input Leakage Current		200	μA	(Note 6)
I_{IL}	Input Leakage Current		−400	μA	(Note 7)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		20	pF	F_C = 1 MHz(8)
C_O	I/O or Output Capacitance		20	pF	F_C = 1 MHz(8)
C_{CLK}	CLK Capacitance		20	pF	F_C = 1 MHz(8)

SEE INTEL FOR DESIGN-IN INFORMATION PRELIMINARY

NOTES:
1. The functional operating temperature range is:
   Intel486 SX CPU, T_{CASE} = 0°C to +85°C.
   Intel487 SX MCP, T_{CASE} = 0°C to +85°C.
   OverDrive Processor—16 MHz, 20 MHz, T_{CASE} = 0°C to +95°C.
   OverDrive Processor—25 MHz, 33 MHz, T_{SINK} = 0°C to +85°C.
2. This parameter is measured at:
   Address, Data, BEn 4.0 mA
   Definition, Control 5.0 mA
3. This parameter is measured at:
   Address, Data, BEn −1.0 mA
   Definition, Control −0.9 mA
4. Typical supply current:
   I_{CC} @ 16 MHz = 340 mA (Normal Operation)
   @ 20 MHz = 450 mA
   @ 25 MHz = 550 mA
   @ 33 MHz = 590 mA
   I_{CCF} @ 16 MHz = 265 mA i486 SX CPU Tri-stated (Floating)
   @ 20 MHz = 375 mA
   @ 25 MHz = 470 mA
   @ 33 MHz = 565 mA
5. This parameter is for inputs without pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
6. This parameter is for inputs with pulldowns and V_{IH} = 2.4V.
7. This parameter is for inputs with pullups and V_{IL} = 0.45V.
8. Not 100% tested.

Table 14.3. Intel486™ SX Microprocessor D.C. Parametric Values (for PQFP package)

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.3	V	
V_{OL}	Output Low Voltage	*	0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current CLK = 16 MHz CLK = 20 MHz CLK = 25 MHz CLK = 33 MHz		 450 500 560 685	 mA	(Note 3)
I_{CCF}	Power Supply Current with Intel486 SX CPU in Power Down Mode		50	mA	(Note 7)
I_{LI}	Input Leakage Current	*	±15	μA	(Note 4)
I_{IH}	Input Leakage Current		200	μA	(Note 5)
I_{IL}	Input Leakage Current		−400	μA	(Note 6)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		10	pF	F_C = 1 MHz(7)
C_O	I/O or Output Capacitance		10	pF	F_C = 1 MHz(7)
C_{CLK}	CLK Capacitance		6	pF	F_C = 1 MHz(7)

PRELIMINARY
SEE INTEL FOR DESIGN-IN
INFORMATION

NOTES:
1. This parameter is measured at:
   Address, Data, BEn 4.0 mA
   Definition, Control 5.0 mA
2. This parameter is measured at:
   Address, Data, BEn −1.0 mA
   Definition, Control −0.9 mA
3. Typical supply current:
   I_{CC} 280 mA @ CLK = 16 MHz (Normal Operation)
   340 mA @ CLK = 20 MHz
   430 mA @ CLK = 25 MHz
   590 mA @ CLK = 33 MHz
4. This parameter is for inputs without pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with pulldowns and V_{IH} = 2.4V.
6. This parameter is for inputs with pullups and V_{IL} = 0.45V.
7. Not 100% tested.

Table 14.4. Intel OverDrive™ Processor D.C. Parametric Values

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current CLK = 16 MHz CLK = 20 MHz CLK = 25 MHz CLK = 33 MHz		 625 775 950 1200	 mA	(Note 3)
I_{LI}	Input Leakage Current		±15	μA	(Note 4)
I_{IH}	Input Leakage Current		200	μA	(Note 5)
I_{IL}	Input Leakage Current		−400	μA	(Note 6)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		20(8)	pF	F_C = 1 MHz(7)
C_O	I/O or Output Capacitance		20(8)	pF	F_C = 1 MHz(7)
C_{CLK}	CLK Capacitance		20(8)	pF	F_C = 1 MHz(7)

PRELIMINARY
FOR DESIGN-IN
INFORMATION
SEE INTEL

NOTES:
1. This parameter is measured at:
   Address, Data, BEn 4.0 mA
   Definition, Control 5.0 mA
2. This parameter is measured at:
   Address, Data, BEn −1.0 mA
   Definition, Control −0.9 mA
3. Typical supply current:
   I_{CC} @ 16 MHz = 525 mA
   @ 20 MHz = 625 mA
   @ 25 MHz = 775 mA
   @ 33 MHz = 975 mA
4. This parameter is for inputs without pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with pulldowns and V_{IH} = 2.4V.
6. This parameter is for inputs with pullups and V_{IL} = 0.45V.
7. Not 100% tested.
8. The Intel487 SX Math CoProcessor is not offered at 33 MHz.
The following are the capacitance specifications for the OverDrive Processor at 33 MHz:
   C_{IN} = 13 pF
   C_O = 17 pF
   C_{CLK} = 15 pF

14.4 A.C. Specifications

The A.C. specifications, given in Tables 14.5, 14.6 and 14.7 consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the rising edge of the CLK signal.

A.C. specifications measurement is defined by Figures 14.1–14.3. Inputs must be driven to the voltage levels indicated by Figure 14.3 when A.C. specifications are measured. Intel486 SX microprocessor/ Intel OverDrive Processor output delays are specified with minimum and maximum limits, measured as shown. The minimum Intel486 SX microprocessor/ Intel OverDrive Processor delay times are hold times provided to external circuitry. Intel486 SX microprocessor/Intel OverDrive Processor input setup and hold times are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct Intel486 SX microprocessor/ Intel OverDrive Processor operation.

Table 14.5. 16 MHz Intel486™ SX Microprocessor/ Intel OverDrive™ Processor A.C. Characteristics

V_{CC} = 5V ±5%; (Note 1); C_L = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	16	MHz		1X Clock
t_1	CLK Period	62.5	125	ns	13.1	
t_{1a}	CLK Period Stability		0.1%			Adjacent Clocks
t_2	CLK High Time	20		ns	13.1	at 2V
t_3	CLK Low Time	20		ns	13.1	at 0.8V
t_4	CLK Fall Time		8	ns	13.1	2V to 0.8V
t_5	CLK Rise Time		8	ns	13.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	26	ns	13.5	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		42	ns	13.6	After Clock Edge(2)
t_8	PCHK# Valid Delay	3	35	ns	13.4	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	35	ns	13.5	
t_9	BLAST#, PLOCK# Float Delay		42	ns	13.6	After Clock Edge(2)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	30	ns	13.5	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		42	ns	13.6	After Clock Edge(2)
t_{12}	EADS# Setup Time	12		ns	13.2	
t_{13}	EADS# Hold Time	3		ns	13.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	12		ns	13.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	13.2	
t_{16}	RDY#, BRDY# Setup Time	12		ns	13.2	
t_{17}	RDY#, BRDY# Hold Time	3		ns	13.2	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	12		ns	13.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	13.2	

PRELIMINARY
FOR DESIGN-IN
SEE INTEL
INFORMATION

Table 14.5. 16 MHz Intel486™ SX Microprocessor/ Intel OverDrive™ Processor A.C. Characteristics (Continued)

V_{CC} = 5V ±5%; (Note 1); C_L = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	14		ns	13.2	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	13.2	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	10		ns	13.2	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	13.2	

NOTE:
1. Intel486 SX CPU, T_{CASE} = 0°C to +85°C
   Intel487 SX MCP, T_{CASE} = 0°C to +85°C
   OverDrive Processor, T_{CASE} = 0°C to +95°C
2. Not 100% tested, guaranteed by design characterization.

Table 14.6. 20 MHz Intel486™ SX Microprocessor/ Intel OverDrive™ Processor A.C. Characteristics

V_{CC} = 5V ±5%; (Note 1); C_L = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	20	MHz		1X Clock
t_1	CLK Period	50	125	ns	13.1	
t_{1a}	CLK Period Stability		0.1%			Adjacent Clocks
t_2	CLK High Time	16		ns	13.1	at 2V
t_3	CLK Low Time	16		ns	13.1	at 0.8V
t_4	CLK Fall Time		6	ns	13.1	2V to 0.8V
t_5	CLK Rise Time		6	ns	13.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	23	ns	13.5	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		37	ns	13.6	After Clock Edge(2)
t_8	PCHK# Valid Delay	3	28	ns	13.4	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	28	ns	13.5	
t_9	BLAST#, PLOCK# Float Delay		37	ns	13.6	After Clock Edge(2)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	26	ns	13.5	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		37	ns	13.6	After Clock Edge(2)
t_{12}	EADS# Setup Time	10		ns	13.2	
t_{13}	EADS# Hold Time	3		ns	13.2	

PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

Table 14.6. 20 MHz Intel486™ SX Microprocessor/ Intel OverDrive™ Processor A.C. Characteristics (Continued)

V_{CC} = 5V ±5%; (Note 1); C_L = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{14}	KEN#, BS16#, BS8# Setup Time	10		ns	13.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	13.2	
t_{16}	RDY#, BRDY# Setup Time	10		ns	13.2	
t_{17}	RDY#, BRDY# Hold Time	3		ns	13.2	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	12		ns	13.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	13.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	12		ns	13.2	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	13.2	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	6		ns	13.2	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	13.2	

NOTE:
1. Intel486 SX CPU, T_{CASE} = 0°C to +85°C
   Intel487 SX MCP, T_{CASE} = 0°C to +85°C
   OverDrive Processor, T_{CASE} = 0°C to +95°C
2. Not 100% tested, guaranteed by design characterization.

Table 14.7. 25 MHz Intel486™ SX Microprocessor/ Intel OverDrive™ Processor A.C. Characteristics

V_{CC} = 5V ±5%; (Note 1); C_L = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	25	MHz		1X Clock
t_1	CLK Period	40	125	ns	13.1	
t_{1a}	CLK Period Stability		0.1%			Adjacent Clocks
t_2	CLK High Time	14		ns	13.1	at 2V
t_3	CLK Low Time	14		ns	13.1	at 0.8V
t_4	CLK Fall Time		4	ns	13.1	2V to 0.8V
t_5	CLK Rise Time		4	ns	13.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	19	ns	13.5	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		28	ns	13.6	After Clock Edge(2)
t_8	PCHK# Valid Delay	3	24	ns	13.4	

SEE INTEL PRELIMINARY FOR DESIGN-IN INFORMATION

Table 14.7. 25 MHz Intel486™ SX Microprocessor/ Intel OverDrive™ Processor A.C. Characteristics (Continued)

V_{CC} = 5V ±5%; (Note 1); C_L = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{8a}	BLAST#, PLOCK# Valid Delay	3	24	ns	13.5	
t_9	BLAST#, PLOCK# Float Delay		28	ns	13.6	After Clock Edge(2)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	20	ns	13.5	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		28	ns	13.6	After Clock Edge(2)
t_{12}	EADS# Setup Time	8		ns	13.2	
t_{13}	EADS# Hold Time	3		ns	13.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	8		ns	13.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	13.2	
t_{16}	RDY#, BRDY# Setup Time	8		ns	13.2	
t_{17}	RDY#, BRDY# Hold Time	3		ns	13.2	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	10		ns	13.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	13.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	10		ns	13.2	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	13.2	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	5		ns	13.2	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	13.2	

PRELIMINARY
INTEL FOR DESIGN-IN
INFORMATION

NOTE:
1. Intel486 SX CPU, T_{CASE} = 0°C to +85°C
   Intel487 SX MCP, T_{CASE} = 0°C to +85°C
   OverDrive Processor, T_{SINK} = 0°C to +85°C
2. Not 100% tested, guaranteed by design characterization.

Table 14.8. 33 MHz Intel486™ SX Microprocessor/Intel OverDrive™ Processor A.C. Characteristics

V_{CC} = 5V ±5%; (Note 1); C_L = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	33	MHz		1X Clock
t_1	CLK Period	30	125	ns	13.1	
t_{1a}	CLK Period Stability		0.1%			Adjacent Clocks
t_2	CLK High Time	11		ns	13.1	at 2V
t_3	CLK Low Time	11		ns	13.1	at 0.8V
t_4	CLK Fall Time		3	ns	13.1	2V to 0.8V
t_5	CLK Rise Time		3	ns	13.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Valid Delay	3	16	ns	13.5	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		20	ns	13.6	After Clock Edge(1)
t_8	PCHK# Valid Delay	3	22	ns	13.4	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	20	ns	13.5	
t_9	BLAST#, PLOCK# Float Delay		20	ns	13.6	After Clock Edge(1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	18	ns	13.5	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		20	ns	13.6	After Clock Edge(1)
t_{12}	EADS# Setup Time	5		ns	13.2	
t_{13}	EADS# Hold Time	3		ns	13.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	5		ns	13.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	13.2	
t_{16}	RDY#, BRDY# Setup Time	5		ns	13.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	13.3	
t_{18}	HOLD, AHOLD Setup Time	6		ns	13.2	
t_{18a}	BOFF# Setup Time	8		ns	13.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	13.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR Setup Time	5		ns	13.2	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR Hold Time	3		ns	13.2	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	5		ns	13.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	13.3	

NOTE:
1. Intel486 SX CPU, T_{CASE} = 0°C to +85°C
   Intel487 SX MCP, T_{CASE} = to +85°C
   OverDrive Processor, T_{SINK} = 0°C to +85°C

Table 14.9. Intel486 SX Microprocessor A.C. Characteristics for Boundary Scan Test Inputs (available in PQFP version only)

V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = 50 pF. All Inputs and Outputs are TTL Level.

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{24}	TCK Frequency		25	MHz		1X Clock
t_{25}	TCK Period	40		ns		(Note 2)
t_{26}	TCK High Time	40		ns		@ 2.0V
t_{27}	TCK Low Time	40		ns		@ 0.8V
t_{28}	TCK Rise Time		4	ns		(Note 1)
t_{29}	TCK Fall Time		4	ns		(Note 1)
t_{30}	TDI, TMS Setup Time	8		ns	13.7	(Note 3)
t_{31}	TDI, TMS Hold Time	10		ns	13.7	(Note 3)
t_{32}	TDO Valid Delay	3	25	ns	13.7	(Note 3)
t_{33}	TDO Float Delay		36	ns		
t_{34}	All Outputs (Non-Test) Valid Delay	3	25	ns	13.7	(Note 3)
t_{35}	All Outputs (Non-Test) Float Delay		36	ns	13.7	(Note 3)
t_{36}	All Inputs (Non-Test) Setup Time	8		ns	13.7	(Note 3)
t_{37}	All Inputs (Non-Test) Hold Time	7		ns	13.7	(Note 3)

NOTES:
1. Rise/Fall times are measured between 0.8V and 2.0V. Rise/Fall times can be relaxed by 1 ns per 10 ns increase in TCK period.
2. TCK period ≥ CLK period.
3. Parameter measured from TCK.

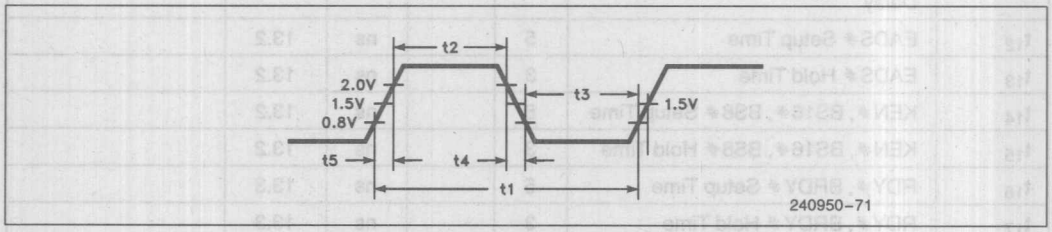


Figure 14.1. CLK Waveforms

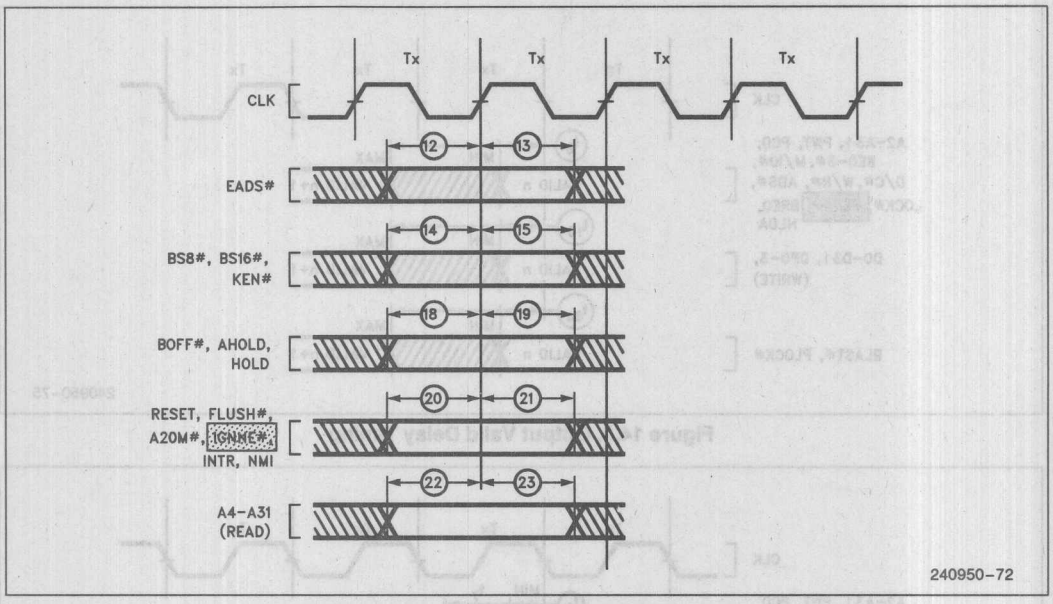


Figure 14.2. Input Setup and Hold Timing

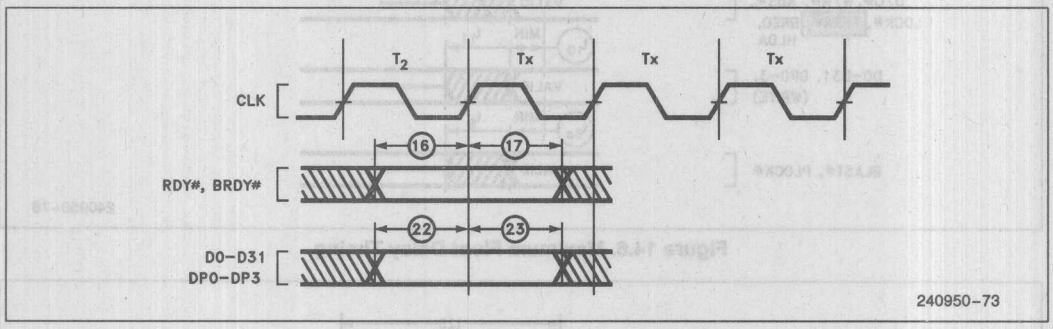


Figure 14.3. Input Setup and Hold Timing

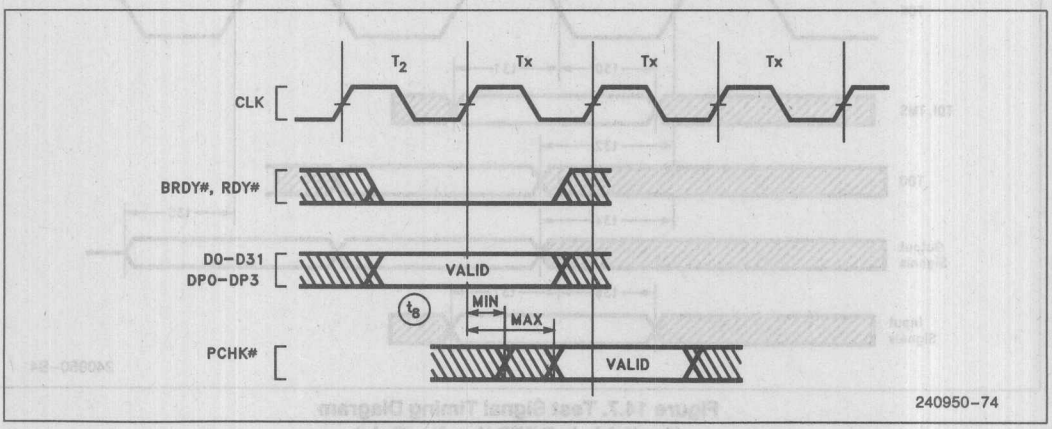


Figure 14.4. PCHK # Valid Delay Timing

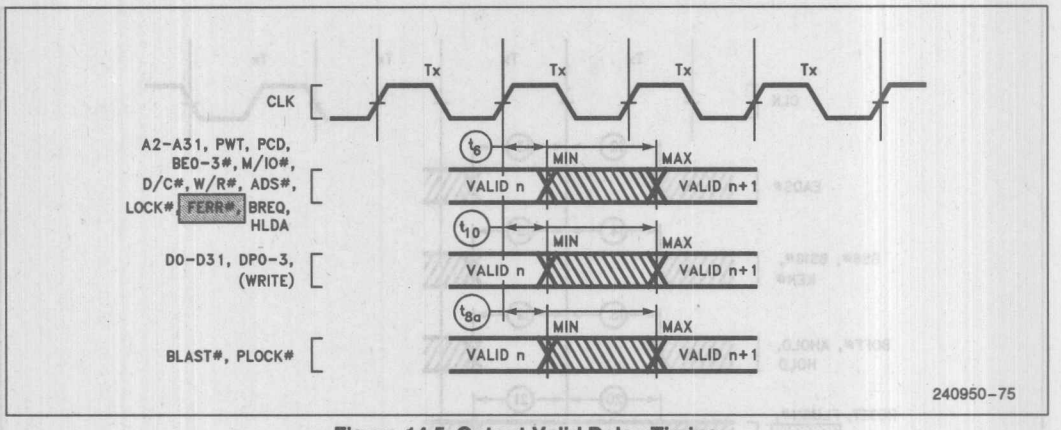


Figure 14.5. Output Valid Delay Timing

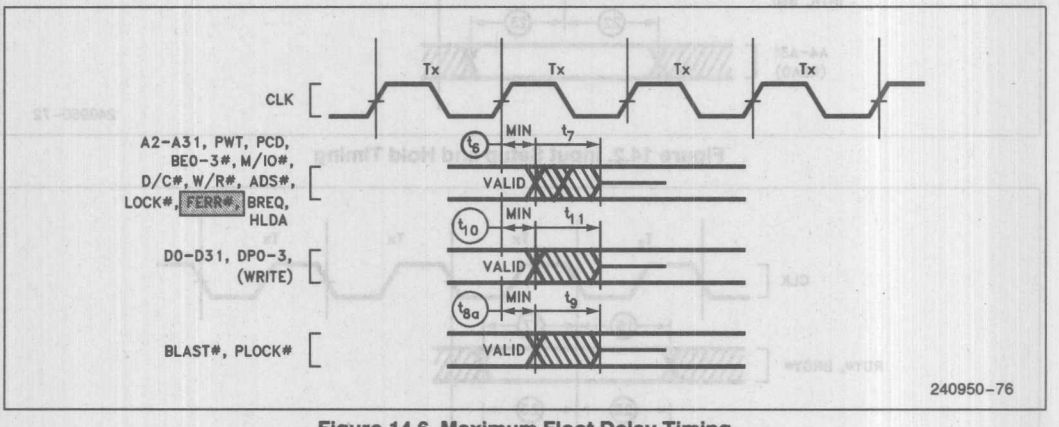


Figure 14.6. Maximum Float Delay Timing

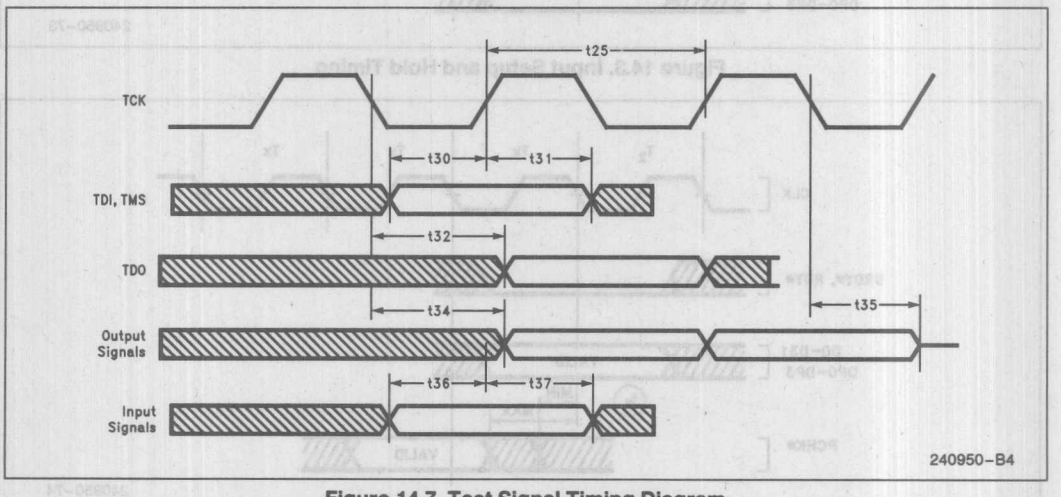
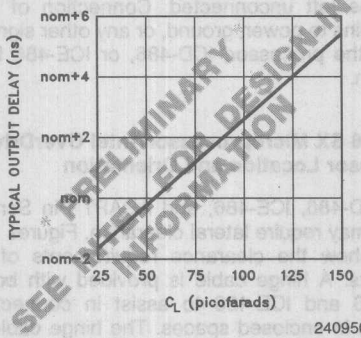


Figure 14.7. Test Signal Timing Diagram
(Available in PQFP Version Only)

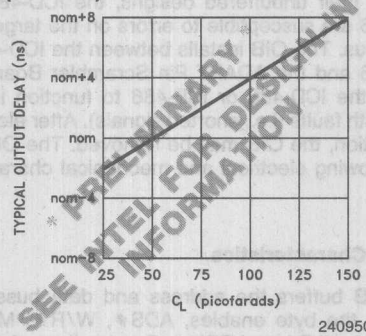
14.4.1 Typical Output Valid Delay versus Load Capacitance Under Worst Case Conditions for Intel486 SX CPU/Intel OverDrive Processor



240950-77

NOTE:
This graph will not be linear outside of the C_L range shown.
nom = nominal value given in A.C. Characteristics table.

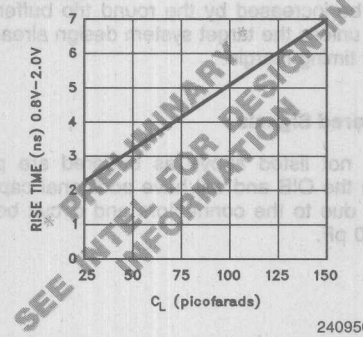
14.4.2 Typical Output Valid Delay versus Load Capacitance Under Worst Case Conditions for Intel486 SX CPU (in PQFP)



240950-B2

NOTE:
This graph will not be linear outside of the C_L range shown.
nom = nominal value given in A.C. Characteristics table.

14.4.3 Typical Output Rise Time versus Load Capacitance Under Worst-Case Conditions



240950-78

NOTE:
This graph will not be linear outside of the C_L range shown.

14.5 Designing for ICD-486 or ICE™-486 Using the ADAPT Pin Scrambler Board for the Intel486™ SX Microprocessor/ Intel OverDrive Processor (Advance Information)

The ICD-486 (In-Circuit Debugger) is a hardware assisted debugger for the Intel486 SX microprocessor/ Intel OverDrive Processor. The ICE-486 (In-Circuit Emulator) is an emulator for the Intel486 SX microprocessor/Intel OverDrive Processor. The ADAPT Pin Scrambler Board is a small circuit board which provides an Intel486 DX socket on top and can be jumpered to plug into an Intel486 SX, Intel487 SX or Intel486 DX socket. This Pin Scrambler Board is used to convert an Intel486 SX or Intel487 SX socket into an Intel486 DX socket which the ICD-486 or ICE-486 can plug into.

To use the ICD-486 or ICE-486, the Intel486 SX microprocessor/Intel OverDrive Processor component must be removed from its socket and replaced with the Pin Scrambler Board with its jumpers set to the Intel486 SX microprocessor/Intel OverDrive Processor position. The ICD-486 or ICE-486 can then be plugged into the ADAPT Pin Scrambler Board. Because of the operating frequency of the Intel486 SX microprocessor/Intel OverDrive Processor systems, there is no buffering of signals between the Intel486 DX CPU in the ICD-486 or ICE-486 and the target system. A direct result of the non-buffered interconnect is that the ICD-486 or ICE-486 shares the address and data bus of the target system. In order for the ICD-486 or ICE-486 to function properly (without the Optional Isolation Board installed), the design of the target system must meet the following restrictions:

1. The bus controller must only enable data transceivers onto the data bus during valid read cycles of the Intel486 SX microprocessor/Intel OverDrive Processor.
2. RDY# cannot be used with BREQ to terminate outstanding bus requests (i.e., when using the ICD-486 or ICE-486, BREQ will be asserted when there is not a corresponding assertion of ADS#).
3. Before another bus master drives the local processor address bus, the other bus master must gain access to the address bus through the use of HOLD/HLDA, AHOLD, or BOFF#.
4. The user system must be able to drive one additional CMOS load (approximately 25 pF) on all signals that go to the Intel486 DX emulation processor.

If the target system does not satisfy these restrictions, the Optional Isolation Board (OIB) should be used to isolate the emulation processor from the target system (see following section).

* PRELIMINARY * SEE INTEL FOR DESIGN-IN INFORMATION

In addition to the above restrictions, the ICD-486, ICE-486 and ADAPT Pin Scrambler Board have several electrical and mechanical characteristics that should be taken into consideration when designing the Intel486 SX microprocessor/Intel OverDrive Processor system.

CAPACITIVE LOADING

ICD-486 adds up to 20 pF to each of the Intel486 SX microprocessor/Intel OverDrive Processor signals. For additional detail, refer to the ICD-486/33 In-Circuit Debugger Fact Sheet, literature number 280872-003.

ICE-486 adds up to 55 pF to the CLK signal, and up to 35 pF to each of the other Intel486 SX microprocessor/Intel OverDrive Processor signals. For additional detail, refer to the i486 In-Circuit Emulator Fact Sheet, literature number 280894-001.

In either case, the ADAPT Pin Scrambler Board adds up to an additional 15 pF to each of the Intel486 SX microprocessor/Intel OverDrive Processor signals.

D.C. LOADING

ICD-486 adds ±15 μA loading to the data bus signals and ± 5 μA loading to the address and control signals. For additional detail, refer to the ICD-486/33 In-Circuit Debugger Fact Sheet, literature number 280872-003.

ICE-486 adds ±15 μA loading to the CLK and data bus signals and ±5 μA loading to the address and control signals. For additional detail, refer to the i486 In-Circuit Emulator Fact Sheet, literature number 280894-003.

Power Requirements

For noise immunity and CMOS latch-up protection the ICD-486 and ICE-486 are powered by the target system through the power and ground pins of the Intel486 SX microprocessor/Intel OverDrive Processor socket.

ICD-486 draws up to 1500 mA including the Intel486 DX emulation processor current.

ICE-486 draws up to 2200 mA including the Intel486 DX emulation processor current.

No Connects

Pins specified as N.C. in the Intel486 SX microprocessor/Intel OverDrive Processor pin description must be left unconnected. Connection of any of these pins to power, ground, or any other signal may cause the processor, ICD-486, or ICE-486 to malfunction.

Intel486 SX Microprocessor/Intel OverDrive Processor Location and Orientation

The ICD-486, ICE-486, and ADAPT Pin Scrambler Board may require lateral clearance. Figures 13.8a–13.8f show the clearance requirements of these products. A hinge cable is provided with both the ICD-486 and ICE-486 to assist in connecting to sockets in enclosed spaces. The hinge cable adds an additional 15 pF of capacitive loading and approximately 0.5 ns of propagation delay to each signal.

OPTIONAL ISOLATION BOARD (OIB)

Due to their unbuffered designs, the ICD-486 and ICE-486 are susceptible to errors on the target system's bus. The OIB installs between the ICD-486 or ICE-486 and the ADAPT Pin Scrambler Board and allows the ICD-486 or ICE-486 to function in systems with faults (i.e., shorted signals). After electrical verification, the OIB may be removed. The OIB has the following electrical and mechanical characteristics:

Buffer Characteristics

The OIB buffers the address and data busses as well as the byte enables, ADS#, W/R#, M/IO#, BLAST#, HLDA and RDY#. For details on A.C. and D.C. loading characteristics, refer to the ICD-486/33 In-Circuit Debugger Fact Sheet, literature number 280872-003, and the i486 In-Circuit Emulator Fact Sheet, literature number 280894-001. To guarantee proper operation with the OIB, the clock period should be increased by the round trip buffer delay (10 ns) unless the target system design already has enough timing margin.

Unbuffered Signals

Signals not listed above as buffered are passed through the OIB and will have additional capacitive loading due to the connectors and circuit board of up to 10 pF.

Power Requirements

The OIB is also powered by the target system through the Intel486 SX microprocessor/Intel OverDrive Processor socket and requires 500 mA in addition to the ICD-486 or ICE-486 requirements.

OIB Clearance Requirements

The OIB requires an extra 0.85″ of vertical clearance in the target system above the ADAPT Pin Scrambler Board.

RELOCATABLE EXPANSION MEMORY (REM) BOARD

Power Requirements

The REM is also powered by the target system through the Intel486 SX microprocessor/Intel OverDrive Processor socket and requires 2100 mA in addition to the ICD-486 or ICE-486 requirements.

REM Clearance Requirements

The REM requires an extra 0.85″ of vertical clearance in the target system above the ADAPT Pin Scrambler Board.

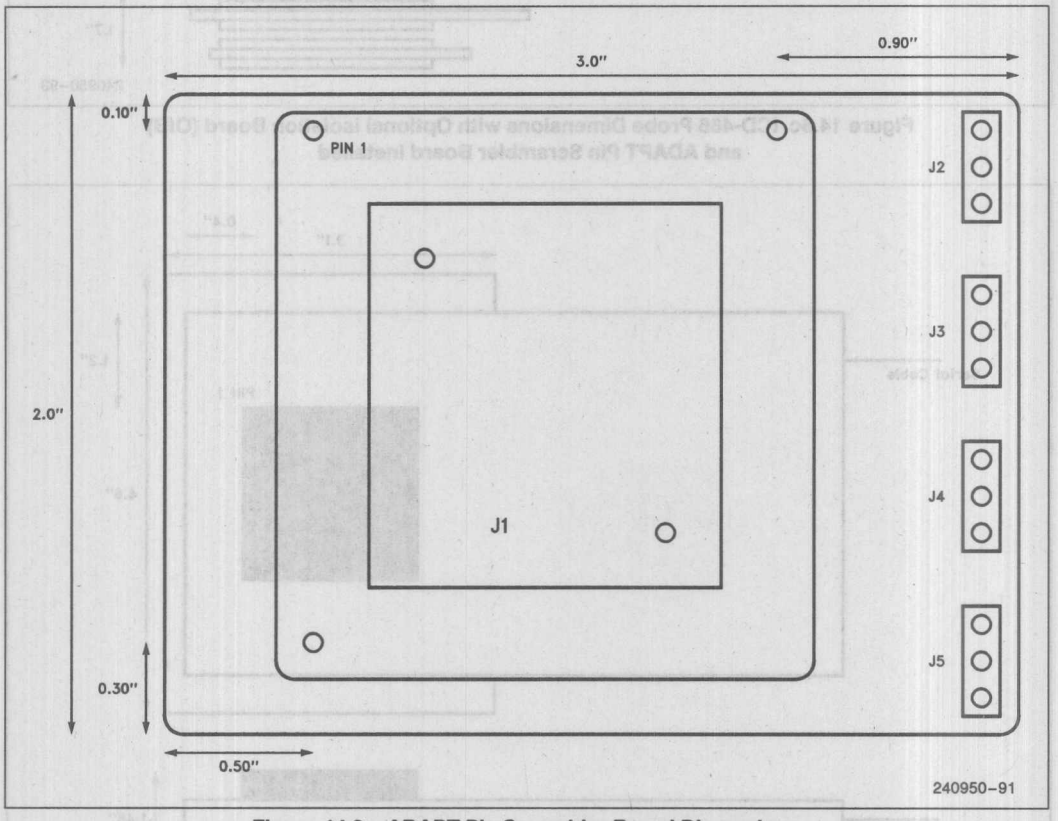


Figure 14.8a. ADAPT Pin Scrambler Board Dimensions

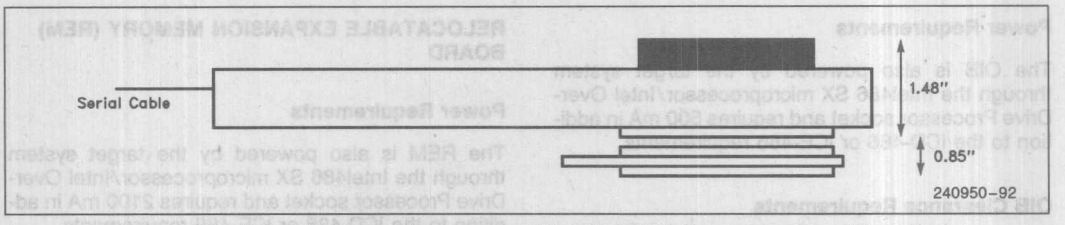


Figure 14.8b. ICD-486 Probe Dimensions with ADAPT Pin Scrambler Board Installed

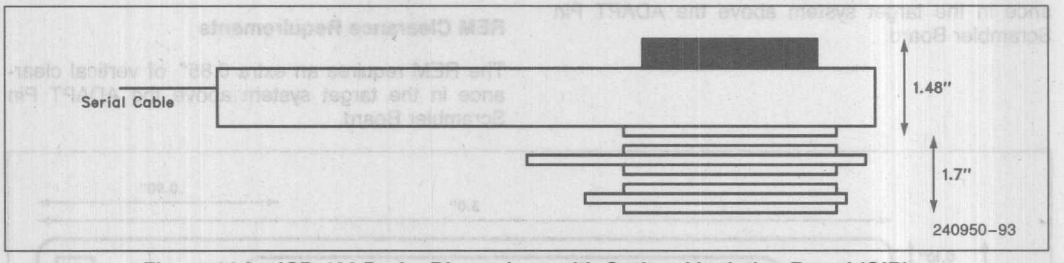


Figure 14.8c. ICD-486 Probe Dimensions with Optional Isolation Board (OIB) and ADAPT Pin Scrambler Board Installed

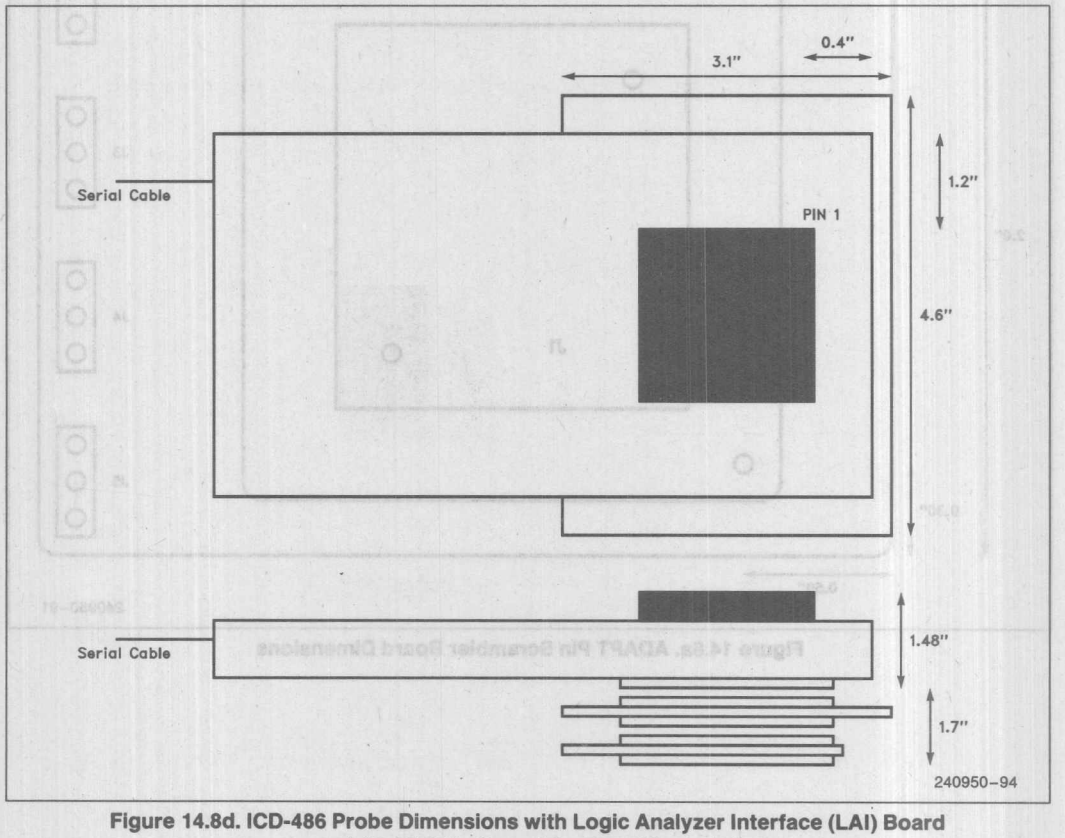


Figure 14.8d. ICD-486 Probe Dimensions with Logic Analyzer Interface (LAI) Board and ADAPT Pin Scrambler Board Installed

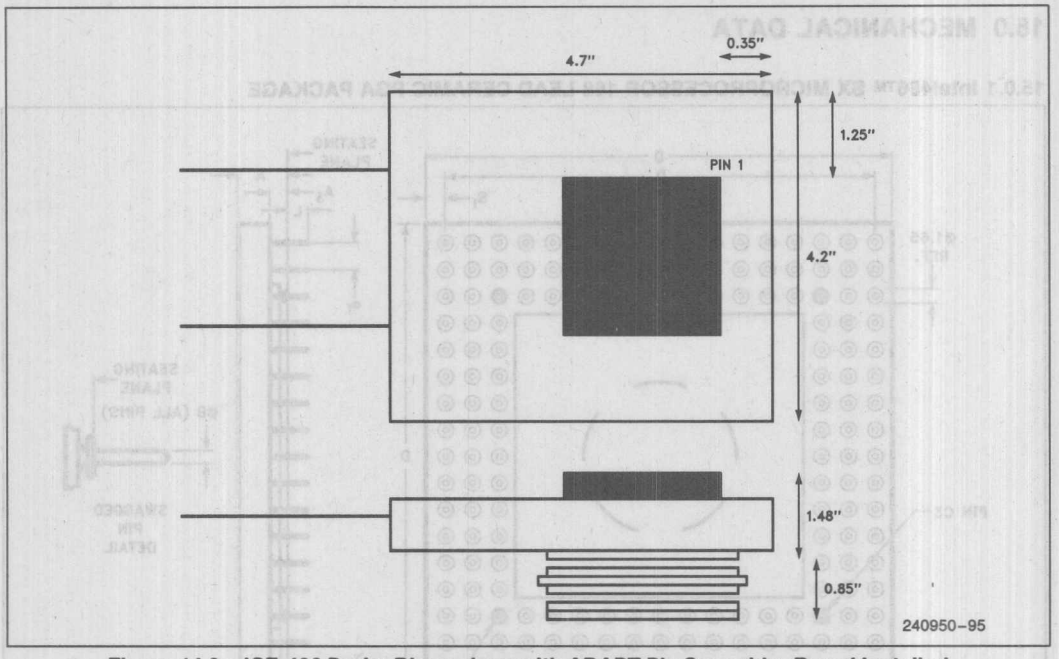


Figure 14.8e. ICE-486 Probe Dimensions with ADAPT Pin Scrambler Board Installed

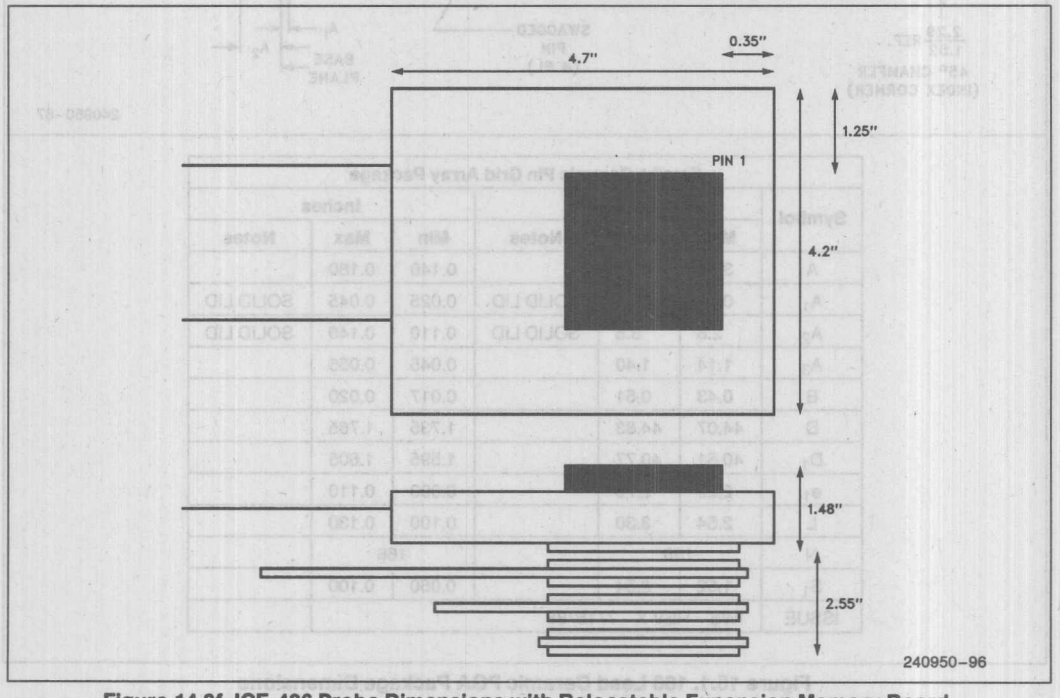
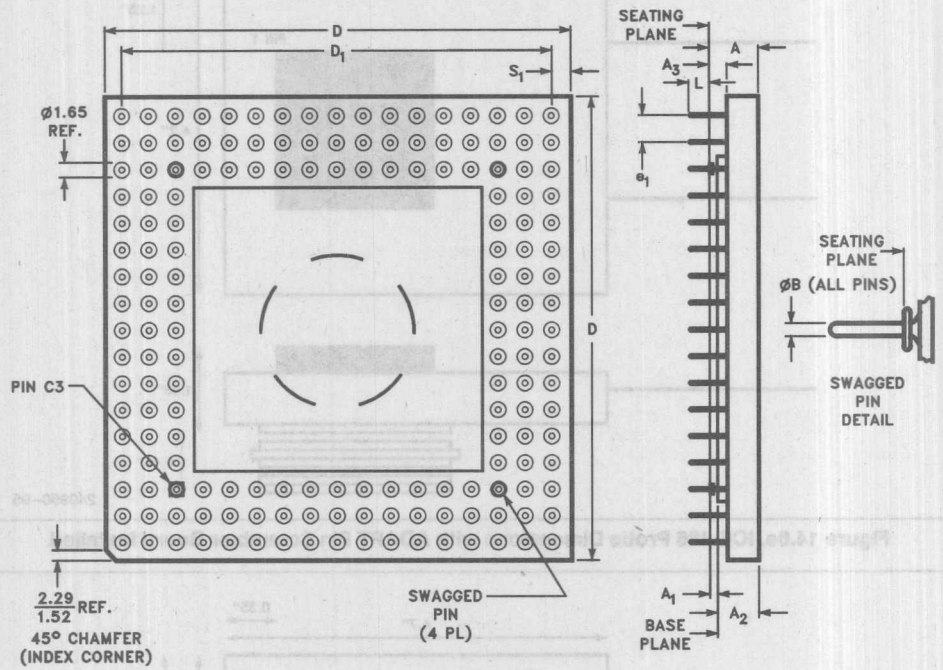


Figure 14.8f. ICE-486 Probe Dimensions with Relocatable Expansion Memory Board, Optional Isolation Board (OIB) and ADAPT Pin Scrambler Board Installed

15.0 MECHANICAL DATA

15.0.1 Intel486™ SX MICROPROCESSOR 168 LEAD CERAMIC PGA PACKAGE



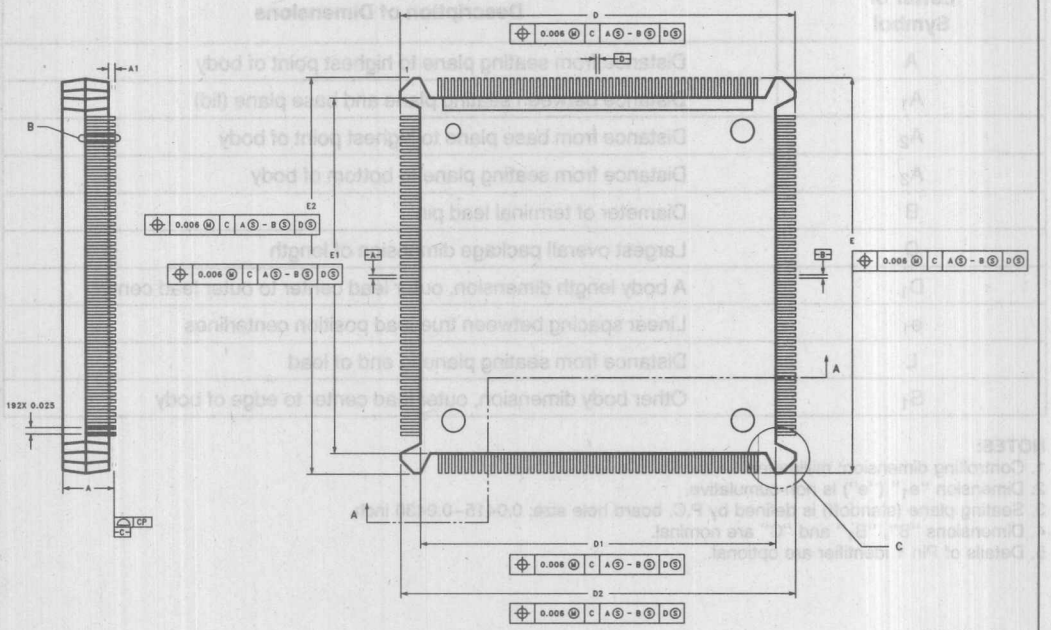
Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		0.140	0.180	
A_1	0.64	1.14	SOLID LID	0.025	0.045	SOLID LID
A_2	2.8	3.5	SOLID LID	0.110	0.140	SOLID LID
A_3	1.14	1.40		0.045	0.055	
B	0.43	0.51		0.017	0.020	
D	44.07	44.83		1.735	1.765	
D_1	40.51	40.77		1.595	1.605	
e_1	2.29	2.79		0.090	0.110	
L	2.54	3.30		0.100	0.130	
N	168			168		
S_1	1.52	2.54		0.060	0.100	
ISSUE	IWS REV X 7/15/88					

Figure 15.1. 168 Lead Ceramic PGA Package Dimensions

Table 15.1 Ceramic PGA Package Dimension Symbols

Letter or Symbol	Description of Dimensions
A	Distance from seating plane to highest point of body
A_1	Distance between seating plane and base plane (lid)
A_2	Distance from base plane to highest point of body
A_3	Distance from seating plane to bottom of body
B	Diameter of terminal lead pin
D	Largest overall package dimension of length
D_1	A body length dimension, outer lead center to outer lead center
e_1	Linear spacing between true lead position centerlines
L	Distance from seating plane to end of lead
S_1	Other body dimension, outer lead center to edge of body

NOTES:
1. Controlling dimension: millimeter.
2. Dimension "e_1" ("e") is non-cumulative.
3. Seating plane (standoff) is defined by P.C. board hole size: 0.0415–0.0430 inch.
4. Dimensions "B", "B_1" and "C" are nominal.
5. Details of Pin 1 identifier are optional.



NOTES:
1. Interpret dimensions and tolerances in accordance with ANSI Y14.5M-1982.
2. Data enclosed in parentheses is for reference only.

Figure 15.2a. Principal Dimensions and Data for 196 Lead PQFP Package

Table 15.2. Symbol List and Dimensions for 196 Lead Plastic Quad Flat Pack Package

Symbol	Description of Dimensions	Min	Max
A	**Package Height:** Distance from the seating plane to the highest point of body.	0.160	0.175
A1	**Standoff:** The distance from the seating plane to the base plane.	0.020	0.035
D, E	**Overall Package Dimension:** Lead tip to lead tip.	1.470	1.485
D1, E1	**Plastic Body Dimension**	1.347	1.353
D2, E2	**Bumper Distance**		
	Without FLASH	1.497	1.503
	With FLASH	1.497	1.510
CP	**Seating Plane Coplanarity**	0.000	0.004

NOTES:
1. All dimensions and tolerances conform to ANSI Y14.5M-1982.
2. Dimensions are in inches.
3. Data enclosed in parentheses is for reference only.

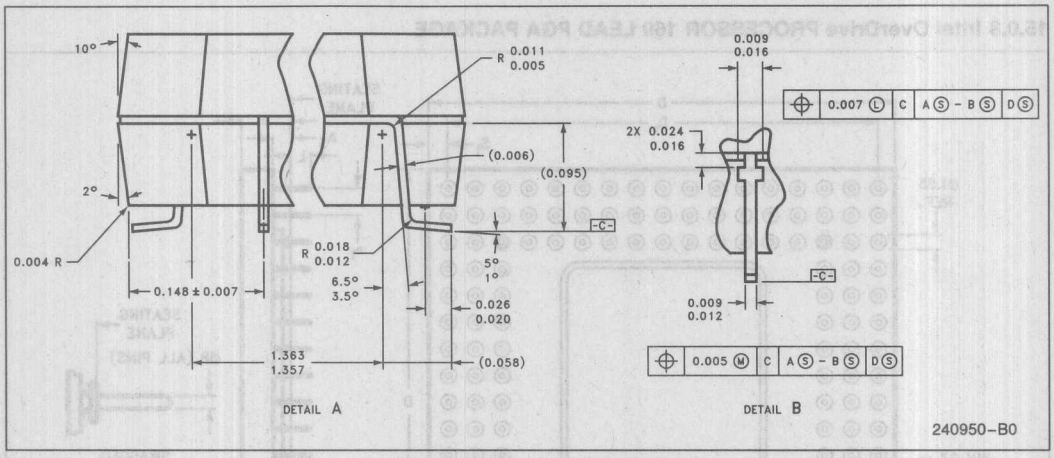


Figure 15.2b. Typical Lead

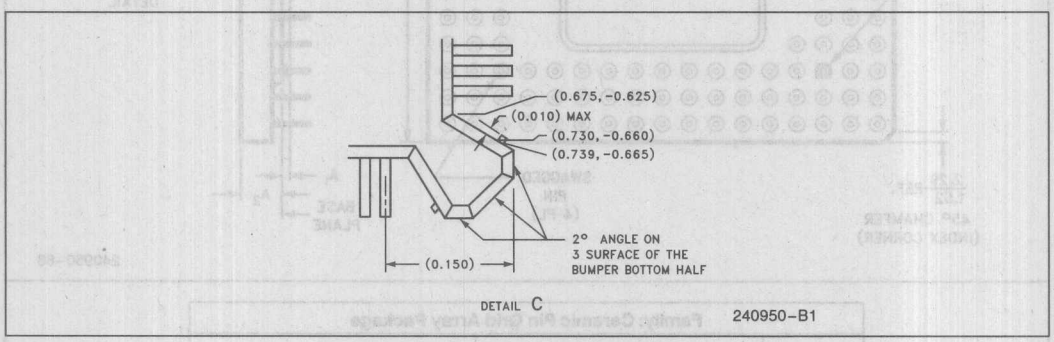
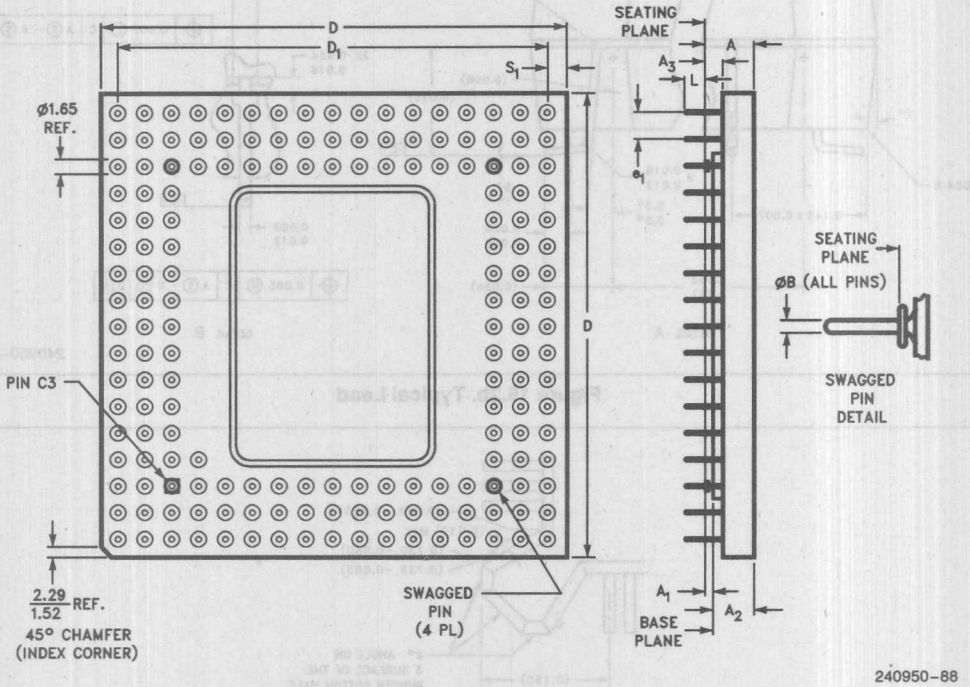


Figure 15.2c. Detail C

15.0.3 Intel OverDrive PROCESSOR 169 LEAD PGA PACKAGE



Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		0.140	0.180	
A_1	0.64	1.14	SOLID LID	0.025	0.045	SOLID LID
A_2	0.23	0.30	SOLID LID	0.110	0.140	SOLID LID
A_3	1.14	1.40		0.045	0.055	
B	0.43	0.51		0.017	0.020	
D	44.07	44.83		1.735	1.765	
D_1	40.51	40.77		1.595	1.605	
e_1	2.29	2.79		0.090	0.110	
L	2.54	3.30		0.100	0.130	
N	169			169		
S_1	1.52	2.54		0.060	0.100	
ISSUE	IWS 1/15/91					

NOTE:
See Table 15.1 for Dimension Symbol descriptions.

Figure 15.3. 169 Lead Ceramic PGA Package Dimensions

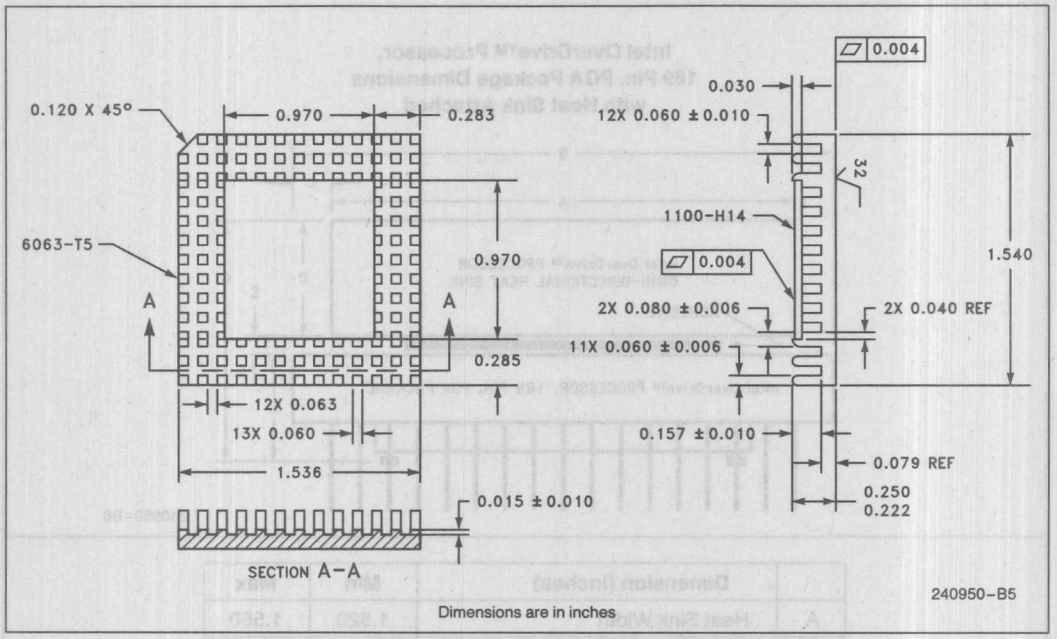
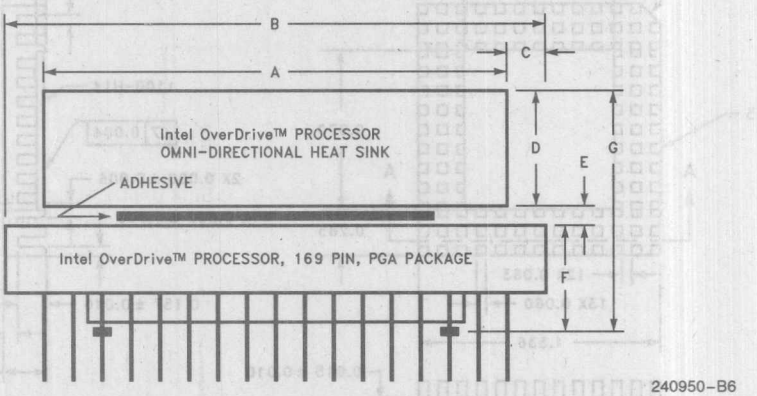


Figure 15.4a. Intel OverDrive™ Processor Heat Sink Dimensions

Intel OverDrive™ Processor, 169 Pin, PGA Package Dimensions with Heat Sink Attached



	Dimension (Inches)	Min	Max
A	Heat Sink Width	1.520	1.550
B	PGA Package Width	1.735	1.765
C	Heat Sink Edge Gap	0.065	0.155
D	Heat Sink Height	0.212	0.260
E	Adhesive Thickness	0.008	0.012
F	Package Height from Stand-Offs	0.140	0.180
G	Total Height from Package Stand-Offs to Top of Heat Sink	0.360	0.452

Figure 15.4b. 169 Pin, PGA Package with Heat Sink

15.1 Package Thermal Specifications

The Intel486 SX microprocessor is specified for operation when T_C (the case temperature) is within the range of 0°C–85°C. T_C may be measured in any environment to determine whether the Intel486 SX microprocessor is within specified operating range. The case temperature, with and without heat sink, is measured using a 0.005″ diameter (AWG #36) thermocouple with a 90° angle adhesive bond at the center of the package top surface, opposite the pins. This is then used to calculate θ_{JC} and θ_{JS}. Figures 15.5 and 15.6 illustrate this methodology. The case temperature for the Intel OverDrive Processor is measured as shown in Figure 15.7.

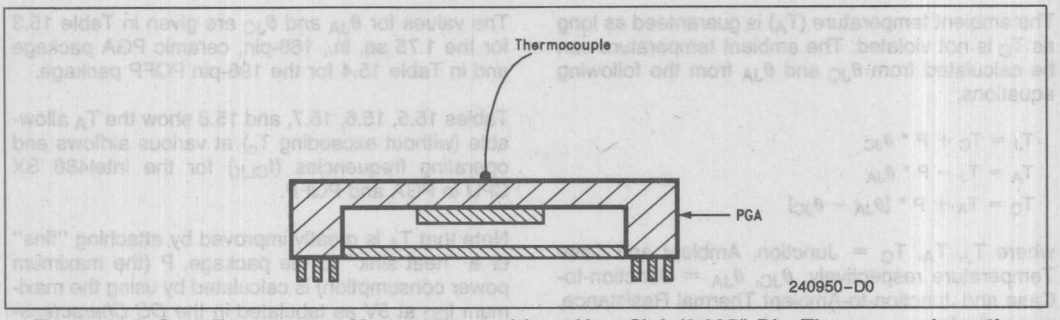


Figure 15.5. Case Temperature Measurement without Heat Sink (0.005" Dia. Thermocouple on the Center of Package Top Surface with a 90° Angle Adhesive Bond).

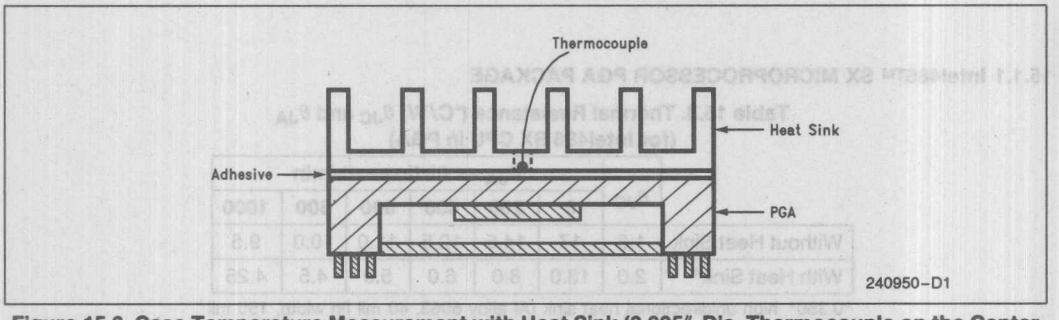


Figure 15.6. Case Temperature Measurement with Heat Sink (0.005" Dia. Thermocouple on the Center of Package Top Surface with a 90° Angle Adhesive Bond through a Hole Drilled at the Heat Sink Base).

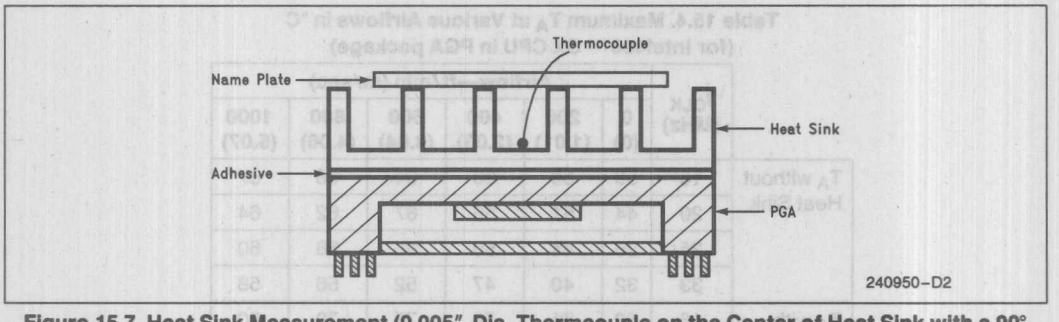


Figure 15.7. Heat Sink Measurement (0.005" Dia. Thermcouple on the Center of Heat Sink with a 90° Angle Adhesive Bond through a Hold Drilled through the Center of the Name Plate.

The ambient temperature (T_A) is guaranteed as long as T_C is not violated. The ambient temperature can be calculated from θ_{JC} and θ_{JA} from the following equations.

$$T_J = T_C + P * \theta_{JC}$$
$$T_A = T_J - P * \theta_{JA}$$
$$T_C = T_A + P * [\theta_{JA} - \theta_{JC}]$$

where T_J, T_A, T_C = Junction, Ambient and Case Temperature respectively. θ_{JC}, θ_{JA} = Junction-to-Case and Junction-to-Ambient Thermal Resistance, respectively.

P = Maximum Power Consumption

The values for θ_{JA} and θ_{JC} are given in Table 15.3 for the 1.75 sq. in., 168-pin, ceramic PGA package and in Table 15.4 for the 196-pin PQFP package.

Tables 15.5, 15.6, 15.7, and 15.8 show the T_A allowable (without exceeding T_c) at various airflows and operating frequencies (f_{CLK}) for the Intel486 SX CPU in PGA and PQFP.

Note that T_A is greatly improved by attaching "fins" or a "heat sink" to the package. P (the maximum power consumption) is calculated by using the maximum I_{CC} at 5V as tabulated in the *DC Characteristics* of Section 14.

15.1.1 Intel486™ SX MICROPROCESSOR PGA PACKAGE

Table 15.3. Thermal Resistance (°C/W) θ_{JC} and θ_{JA} (for Intel486 SX CPU in PGA)

	θ_{JC}	θ_{JA} vs Airflow—ft/min					
		0	200	400	600	800	1000
Without Heat Sink	1.5	17	14.5	12.5	11.0	10.0	9.5
With Heat Sink*	2.0	13.0	8.0	6.0	5.0	4.5	4.25

*0.350" high unidirectional heat sink (Al alloy 6063, 40 mil fin width, 155 mil center-to-center fin spacing). See Figure 15.6 for details on measurement methodology.

Table 15.4. Maximum T_A at Various Airflows in °C (for Intel486™ SX CPU in PGA package)

	f_{CLK} (MHz)	Airflow—ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
T_A without Heat Sink	16	50	56	60	64	66	67
	20	44	51	56	67	62	64
	25	36	44	50	55	58	60
	33	32	40	47	52	56	58
T_A with Heat Sink	16	60	71	76	78	79	80
	20	56	69	74	77	78	79
	25	50	66	72	76	77	78
	33	47	64	71	75	76	77

15.1.2 Intel486™ SX MICROPROCESSOR PQFP PACKAGE

Table 15.5. Thermal Resistance (°C/W) θ_{JC} and θ_{JA} (for Intel486 SX CPU (16, 20, and 25 MHz) in PQFP Package)

	θ_{JC}	θ_{JA} vs Airflow—ft/min			
		0	200	400	600
Without Heat Sink	3.5	22.5	19.0	16.0	14.0
With Heat Sink*	6.0	18.0	12.5	10.0	9.5

*Uni-Directional Heat Sink, H = 0.35″

Table 15.6. Thermal Resistance (°C/W) θ_{JC} and θ_{JA} for the 33 MHz Intel486 SX Microprocessor in PQFP with Enhanced MM Package

	θ_{JC}	θ_{JA} vs Airflow—ft/min			
		0	200	400	600
Without Heat Sink	3.5	20.3	16.5	13.9	12.5
With Heat Sink*	6.0	16.7	10.4	8.3	7.8

*1.35″ sq. 0.350″ high omni-directional pin Al heat sink with 0.050″ pin width, 0.143″ pin-to-pin center spacing and 0.100″ base thickness.

Table 15.7. Maximum T_A at Various Airflows in °C (for Intel486 SX CPU in PQFP Package)

	f_{CLK} (MHz)	Airflow—ft/min (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
T_A without Heat Sink	16	42	50	57	61
	20	37	46	54	59
	25	32	42	50	56
T_A with Heat Sink	16	58	70	76	77
	20	55	69	75	76
	25	51	67	74	75

Table 15.8. Maximum T_A at Various Airflows in °C (for the 33 MHz Intel486 SX Microprocessor in PQFP with Enhanced MM Package)

	f_{CLK} (MHz)	Airflow—ft/min (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
T_A without Heat Sink	33	27	40	49	54
T_A with Heat Sink	33	48	70	77	79

15.1.3 Intel OverDrive Processor

The methodology for calculating the heat dissipation for the 20 MHz Intel OverDrive Processor (without a heat sink) and the 25 MHz, 33 MHz Intel OverDrive Processor (with a heat sink) are identical except that the reference point for measuring the product temperature is different. The 20 MHz Intel OverDrive Processor specifies T_{case} (the temperature at the outside center of the PGA package, opposite the pins) and θ_{JC} (the thermal resistance from the silicon junction to the package case) but the 25 MHz, 33 MHz Intel OverDrive Processor specifies T_{sink} (the temperature at the outside center base of the heat sink, not on the heat sink marking plate or cooling posts) and θ_{JS} (the thermal resistance from the silicon junction to the heat sink base). The relationships between temperature, thermal resistance and power are shown in the following equations:

$$T_{case} = T_{ambient} + (P_{max} * \theta_{CA}) \text{ and}$$

$$T_{sink} = T_{ambient} + (P_{max} * \theta_{SA})$$

where,

$T_{ambient}$ = Ambient Temperature

P_{max} = Power ($I_{CC} * V_{CC}$),

$\theta_{CA} = \theta_{JA} - \theta_{JC}$,

$\theta_{SA} = \theta_{JA} - \theta_{JS}$.

The 20 MHz Intel OverDrive Processor, for 16 MHz and 20 MHz i486 SX CPU systems, is supplied without a heat sink. Table 15.9 contains the thermal resistance values for the Intel OverDrive Processor without heat sink. The thermal resistance values for the Intel OverDrive Processor with attached heat sink (25 MHz, 33 MHz versions) are shown in Table 15.10.

Table 15.9 Thermal Resistance for the Intel OverDrive™ Processor without Heat Sink

θ_{JC} = 2.0°C/W	Airflow (ft/min, LFM)				
	0	200	400	600	800
θ_{JA}(°C/W)	19.0*	16.0	12.5	11.0	10.0

NOTE:
*The thermal resistance from junction to ambient (θ_{JA}) in static air is actually a linear function of power dissipation. The value shown in the table (19.0°C/W) represents the worst case expected value which is derived from a power dissipation of 2.9W. The maximum expected power dissipation of 4W yields a θ_{JA} value of 18.5°C/W.

Table 15.10. Thermal Resistance for the Intel OverDrive™ Processor with Attached Heat Sink

θ_{JS} = 2.5°C/W	Airflow (ft/min, LFM)				
	0	200	400	600	800
θ_{JA}(°C/W)	14.0*	10.0	7.5	6.2	5.7

NOTE:
*The thermal resistance from junction to ambient (θ_{JA}) in static air is actually a linear function of power dissipation. The value shown in the table (14.0°C/W) represents the worst case expected value which is derived from a power dissipation of 2.9W. The maximum expected power dissipation of 6W yields a θ_{JA} value of 13.1°C/W.

Table 15.11 T_A (°C) for the OverDrive™ Processor

	f_{CLK} (MHz)	Airflow—Linear ft/min (m/sec)				
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
OverDrive Processor without Heat Sink(1)	16	42	51	62	67	70
	20	29	41	54	60	64
OverDrive Processor with Heat Sink(2)	25	30	49	61	67	70
	33	16	40	55	63	66

NOTES:
1. The 20 MHz OverDrive Processor does not have a heat sink.
2. The 25 MHz and 33 MHz Overrive Processors have a heat sink attached.

16.0 LOW POWER Intel486™ SX MICROPROCESSOR

Low Power Intel486™ SX CPU/ Intel487™ SX MCP

- Lower Power Dissipation
  - Dynamic Frequency Scalability
  - I_{CC}(max) Reduced to 150 mA at 2 MHz
  - Improved V_{CC} Rating (±10%)
- Binary Compatible with Large Software Base
  - MS-DOS, OS/2, Windows
  - UNIX System V/386
  - iRMX, iRMK Kernels
- High Integration Enables On-Chip
  - 8 KByte Code and Data Cache
  - Floating Point Unit on the Intel487 SX Math CoProcessor
  - Paged, Virtual Memory Management
- Easy to Use
  - Built-In Self Test
  - Hardware Debugging Support
  - Intel Software Support
  - Extensive Third Party Software Support
- 168-Lead Pin Grid Array for Intel486 SX Microprocessor
- 196-Lead Plastic Quad Flat Package for Intel486 SX Microprocessor
- 169-Pin Grid Array Package for Intel487 SX Math CoProcessor
- High Performance Design
  - Intel486 One Clock Instruction Core
  - 16/20/25 MHz Operation for Intel486 SX
  - 64 MByte/Sec Burst Bus
  - CHMOS IV Process Technology
  - Dynamic Bus Sizing for 8-, 16- and 32-Bit Buses
- Complete 32-Bit Architecture
  - Address and Data Buses
  - Registers
  - 8-, 16- and 32-Bit Data Types
- Multiprocessor Support
  - Multiprocessor Instructions
  - Cache Consistency Protocols
  - Support for Second Level Cache

This section describes the Low Power Intel486 SX microprocessor. The Intel487 Math CoProcessor will support the low power Intel486 SX microprocessor as an optional upgrade available through the retail channel.

The Low Power Intel486 family microprocessors meet today's need for high performance portables. Their combination of special features like dynamic frequency scaling, lower minimum frequency, improved V_{CC} operation and high integration contribute significantly to lower power dissipation and meet the needs of portable computing.

The Low Power capability is achieved by operating the Intel486 microprocessor in the 2X mode. The frequency can be varied dynamically between maximum to minimum as needed. The frequency change does not affect contents of the registers and data integrity is maintained. Power dissipation is reduced significantly at 2 MHz where I_{CC} is only 150 mA compared to 600 mA at 20 MHz.

The Low Power Intel486 microprocessors are 100-percent compatible with all versions of the Intel386 microprocessor family, assuring compatibility with the more than $40 billion software base of MS-DOS, Windows, OS/2 and UNIX/System operating system applications. The Low Power Intel486 microprocessor integrates the same RISC-technology, one clock per instruction integer core, on-chip cache, and memory management unit as the standard Intel486 microprocessor.

The Intel487 Math CoProcessor provides optional math upgrade capability for the Intel486 SX microprocessor and supports low power operation; providing end-users increased floating point performance for more than 2100 software packages that were designed to use Intel Math CoProcessors. Note that the Intel OverDrive Processor does not work in systems based on the Low Power Intel486 CPU.

The following section on the Low Power Intel 486 SX microprocessor contains information specific to the Low Power device only. All data not defined are located in the appropriate sections of this data sheet unless specified otherwise.

16.1 Introduction

The Low Power Intel486 microprocessor brings Intel486 technology and performance to the portable computer market. The low power capability is achieved by a frequency scalability feature during normal operation. The operating frequency can be brought down dynamically resulting in lower power supply current (I_{CC}). This results in minimal power dissipation which ensures a longer battery life.

The Low Power Intel486 microprocessor integrates the same RISC-technology, one clock per instruction integer core, on-chip cache, and memory management unit as the standard Intel486 microprocessor.

The Low Power Intel486 microprocessor has the following special features:

- **Frequency Scalability**—This is achieved by operating the Intel486 microprocessor in the 2X clock mode. The frequency can be varied dynamically from maximum back to minimum or vice versa. The frequency change does not affect the register content of the CPU, thus data integrity is maintained.
- **Lower Minimum Frequency**—The Low Power Intel486 microprocessor can be operated at a minimum frequency of 2 MHz, at which I_{CC}(max) is only 150 mA, compared to an I_{CC}(max) of 600 mA at 20 MHz operation. The power dissipation is thus drastically reduced ensuring a longer battery life.
- **Improved V_{CC} Operation**—The Low Power Intel486 microprocessor has an improved V_{CC} rating of ±10%. Again this feature makes it extremely attractive to portable battery powered applications.

The above three features ensure power savings for portable computer systems resulting in prolonged battery life.

Besides these special features, the Low Power Intel486 microprocessor has an identical feature set to the standard Intel486 CPU. This includes:

- **Binary Compatibility**—The Low Power Intel486 CPU is binary compatible with the 8086, 8088, 80186, 80286, i386™ SX, i386™ DX, Intel486™ SX and Intel486™ DX CPUs.
- **Full 32-Bit Integer Processor**—The Low Power Intel486 CPU performs a complete set of arithmetic and logical operations on 8-, 16-, and 32-bit data types using a full-width ALU and eight general-purpose registers.
- **Separate 32-Bit Address and Data Paths**—Four gigabytes of physical memory can be addressed directly.
- **Single-Cycle Execution**—Many instructions execute in a single clock cycle.
- **On-Chip Floating Point Unit**—This is available on the Intel486 DX CPU. The 32-, 64-, and 80-bit formats specified in IEEE standard 754 are supported. The unit is binary compatible with the 8087, 80287, i387™, i387™ SX, and Intel OverDrive Processor and the Intel486™ CPU.
- **On-Chip Memory Management Unit**—Address-management and memory-space protection mechanisms maintain the integrity of memory. This is necessary in multitasking and virtual-memory environments, like those implemented by the UNIX and OS/2 operating systems. Both memory segmentation and paging are supported.
- **On-Chip Cache with Cache Consistency Support**—The internal write-through cache can hold 8 KBytes of data or instructions. Cache hits are as fast as read accesses to a processor register. Bus activity is tracked to detect alterations in the memory which internal cache represents. The internal cache can be invalidated or flushed, so that an external cache controller can maintain cache consistency in multi-processor environments.
- **External Cache Control**—Write-back and flush controls over an external cache are provided so that the processor can maintain cache consistency in multi-processor environments.
- **Instruction Pipelining**—The fetching, decoding, execution and address translation of instructions are overlaped within the Low Power Intel486 microprocessor. This results in a continuous execution rate of one clock cycle per instruction, for most instructions.
- **Burst Cycles**—Burst transfers allow a new doubleword to be read from memory each clock cycle. With this capability the internal cache and instruction prefetch buffer can be filled very rapidly.
- **Write Buffers**—The processor contains write buffers to enhance the performance of consecutive writes to memory. The Low Power Intel486 CPU can continue operations internally after a write, without waiting for the write to be executed on the external bus.
- **Bus Backoff**—If another bus master needs control of the bus during a Low Power Intel486 microprocessor initiated bus cycle, the Low Power Intel486 microprocessor will float its bus signals, then restart its cycle when the bus becomes available again.
- **Instruction Restart**—Programs can continue execution following an exception generated by an unsuccessful attempt to access memory. This feature is important for supporting demand-paged virtual memory applications.
- **Dynamic Bus Sizing**—External controllers can dynamically alter the effective width of the data bus. Bus widths of 8, 16 or 32 bits can be used.

16.2 Pinout

The Low Power Intel486™ SX microprocessor pinout follows the same definition as the Intel486™ SX microprocessor given in Section 1.0, except for the pins listed in Table 16.1.

Table 16.1

i486 SX Microprocessor	Low Power i486 Microprocessor	Pin # PGA	Pin # PQFP
CLK	CLK2	C3	123
NC	CLKSEL	A3	127

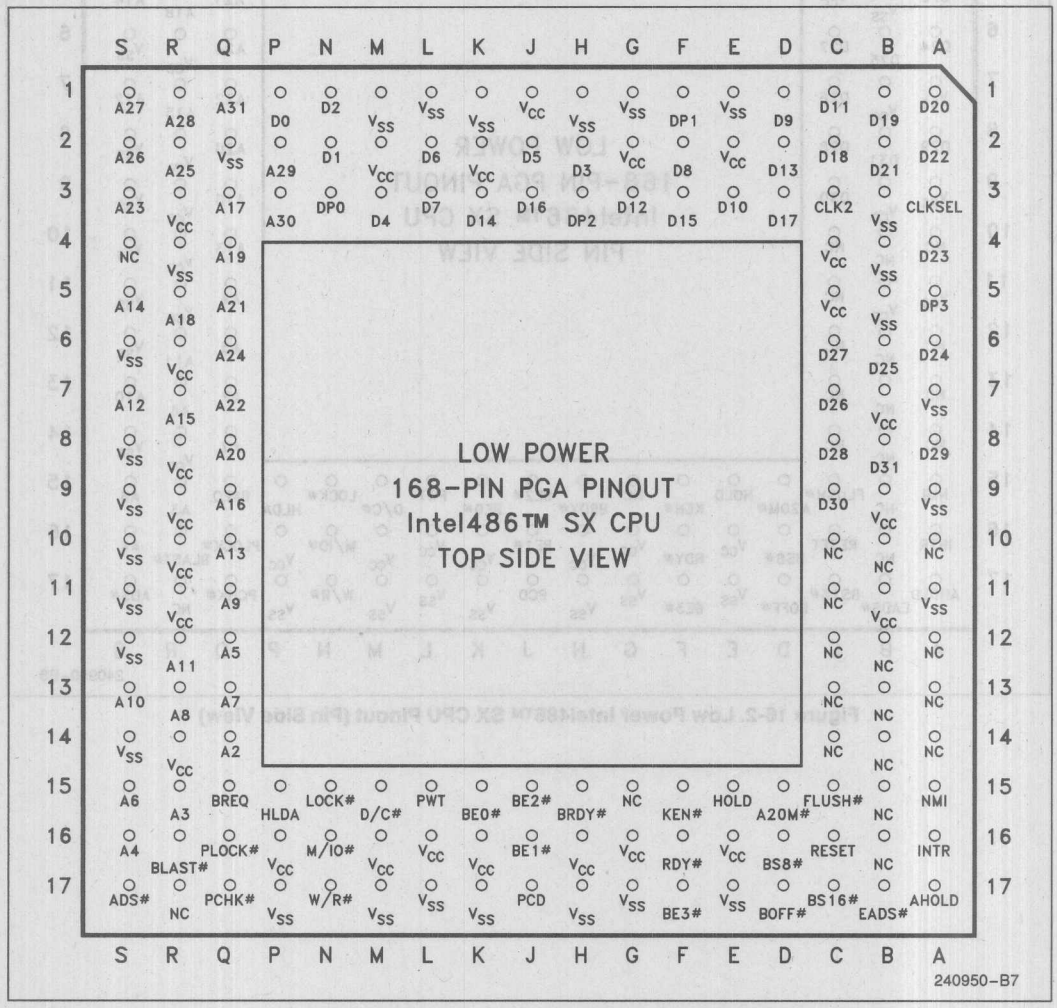


Figure 16-1. Low Power Intel486™ SX CPU Pinout (Top Side View)

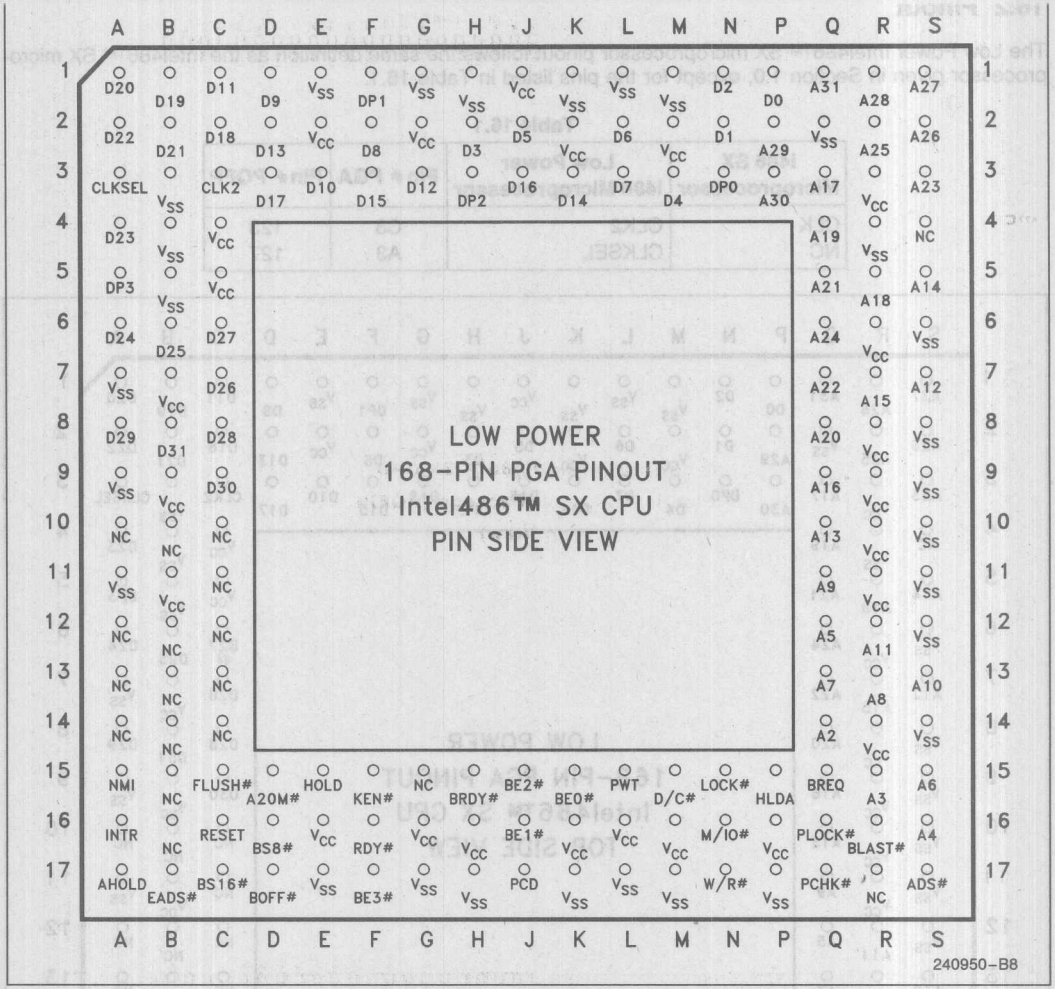


Figure 16-2. Low Power Intel486™ SX CPU Pinout (Pin Side View)



Figure 16-3. Low Power Intel486™ SX CPU 196-Lead PQFP Pinout

16.3 Pin Cross Reference (Intel486™ PGA Version)

Address		Data		Control		N/C	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	A10	B7	A7
A_3	R15	D_1	N2	ADS#	S17	A12	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	A13	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	A14	C4	B3
A_6	S15	D_4	M3	BE1#	J16	B10	C5	B4
A_7	Q13	D_5	J2	BE2#	J15	B12	E2	B5
A_8	R13	D_6	L2	BE3#	F17	B13	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16	B14	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17	B16	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15	C10	H16	G17
A_{12}	S7	D_{10}	E3	BREQ	Q15	C11	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16	C12	K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17	C13	K16	K1
A_{15}	R7	D_{13}	D2	CLK2	C3	G15	L16	K17
A_{16}	Q9	D_{14}	K3	CLKSEL	A3	R17	M2	L1
A_{17}	Q3	D_{15}	F3	D/C#	M15	S4	M16	L17
A_{18}	R5	D_{16}	J3	DP0	N3		P16	M1
A_{19}	Q4	D_{17}	D3	DP1	F1		R3	M17
A_{20}	Q8	D_{18}	C2	DP2	H3		R6	P17
A_{21}	Q5	D_{19}	B1	DP3	A5		R8	Q2
A_{22}	Q7	D_{20}	A1	EADS#	B17		R9	R4
A_{23}	S3	D_{21}	B2	FERR#	C14		R10	S6
A_{24}	Q6	D_{22}	A2	FLUSH#	C15		R11	S8
A_{25}	R2	D_{23}	A4	HLDA	P15		R14	S9
A_{26}	S2	D_{24}	A6	HOLD	E15			S10
A_{27}	S1	D_{25}	B6	IGNNE#	A15			S11
A_{28}	R1	D_{26}	C7	INTR	A16			S12
A_{29}	P2	D_{27}	C6	KEN#	F15			S14
A_{30}	P3	D_{28}	C8	LOCK#	N15			
A_{31}	Q1	D_{29}	A8	M/IO#	N16			
		D_{30}	C9	NMI	B15			
		D_{31}	B8	PCD	J17			
				PCHK#	Q17			
				PWT	L15			
				PLOCK#	Q16			
				RDY#	F16			
				RESET	C16			
				W/R#	N17			

16.4 Pin Cross Reference by Signal Type (Intel486™ SX CPU) (PQFP Version)

Address		Data		Control		N/C	V_{CC}	V_{SS}
A_2	146	D_0	17	A20M#	104	15	6	1
A_3	150	D_1	18	ADS#	145	34	19	11
A_4	152	D_2	20	AHOLD	129	52	24	21
A_5	152	D_3	23	BE0#	117	56	28	22
A_6	158	D_4	25	BE1#	116	60	36	33
A_7	159	D_5	26	BE2#	115	64	49	40
A_8	161	D_6	27	BE3#	113	68	54	50
A_9	163	D_7	29	BLAST#	144	72	62	58
A_{10}	165	D_8	31	BOFF#	137	73	70	66
A_{11}	172	D_9	32	BRDY#	138	75	84	86
A_{12}	174	D_{10}	35	BREQ	118	76	93	95
A_{13}	176	D_{11}	37	BS8#	135	77	98	96
A_{14}	178	D_{12}	38	BS16#	136	78	107	99
A_{15}	180	D_{13}	39	CLK2	123	79	112	109
A_{16}	181	D_{14}	41	CLKSEL	127	81	119	114
A_{17}	183	D_{15}	42	D/C#	110	82	125	121
A_{18}	189	D_{16}	44	DP0	16	83	131	126
A_{19}	191	D_{17}	45	DP1	30	85	147	141
A_{20}	193	D_{18}	46	DP2	43	87	164	148
A_{21}	2	D_{19}	47	DP3	57	88	170	167
A_{22}	3	D_{20}	48	EADS#	105	89	175	168
A_{23}	4	D_{21}	51	FLUSH#	102	90	179	177
A_{24}	5	D_{22}	53	HLDA	122	91	184	182
A_{25}	7	D_{23}	55	HOLD	130	92	196	194
A_{26}	8	D_{24}	59	INTR	101	94		
A_{27}	9	D_{25}	61	KEN#	132	97		
A_{28}	10	D_{26}	63	LOCK#	142	124		
A_{29}	12	D_{27}	65	M/IO#	111	134		
A_{30}	13	D_{28}	67	NMI	100	140		
A_{31}	14	D_{29}	69	PCD	106	149		
		D_{30}	71	PCHK#	139	151		
		D_{31}	74	PWT	108	153		
				PLOCK#	143	155		
				RDY#	133	157		
				RESET	103	160		
				TCK	128	162		
				TDI	185	166		
				TDO	80	166		
				TMS	187	169		
				UP#	156	171		
				W/R#	120	173		
						186		
						188		
						190		
						192		
						195		

16.5 Pin Description

All pin descriptions for the Low Power Intel486 SX microprocessor follow the same definition as the Intel486 SX microprocessor with the exception of those listed in Table 16.2.

Table 16.2

Symbol	Type	Name and Function
CLK2	I	**CLK2** provides the fundamental timing for the Low Power Intel486 SX microprocessor. This is twice the internal frequency of the CPU.
CLKSEL	I	**CLOCK SELECT** pin selects the 2X mode required for the Low Power Intel486 SX CPU. A well defined pulse on this pin establishes the phase relationship of the 2X clock. With the exception of a pulse during cold reset, this pin should be driven low at all times and must be free of spikes or glitches.

OUTPUT PINS

Table 16.3 lists all the output pins, indicating their active level, and when they are floated.

Table 16.3. Output Pins

Name	Active Level	When Floated
BREQ	HIGH	
HLDA	HIGH	
BE0#–BE3#	LOW	Bus Hold
PWT, PCD	HIGH	Bus Hold
W/R#, D/C#, M/IO#	HIGH/LOW	Bus Hold
LOCK#	LOW	Bus Hold
PLOCK#	LOW	Bus Hold
ADS#	LOW	Bus Hold
BLAST#	LOW	Bus Hold
PCHK#	LOW	
FERR#*	LOW	
A2-A3	HIGH	Bus, Address Hold

*Present on Intel486 DX CPU Only

INPUT PINS

Table 16.4 lists all input pins, indicating their active level, and whether they are synchronous or asynchronous inputs.

Table 16.4. Input Pins

Name	Active Level	Synchronous/ Asynchronous
CLK2		
CLKSEL		
RESET	HIGH	Asynchronous
HOLD	HIGH	Synchronous
AHOLD	HIGH	Synchronous
EADS#	LOW	Synchronous
BOFF#	LOW	Synchronous
FLUSH#	LOW	Asynchronous
A20M#	LOW	Asynchronous
BS16#, BS8#	LOW	Synchronous
KEN#	LOW	Synchronous
RDY#	LOW	Synchronous
BRDY#	LOW	Synchronous
INTR	HIGH	Asynchronous
NMI	HIGH	Asynchronous
IGNNE#*(1)	LOW	Asynchronous
UP#(2)	LOW	Asynchronous

NOTES:
1. The IGNNE# pin is present on the Intel486 DX CPU and Intel487 SX MCP only.
2. The UP# pin is present on the Intel486 SX CPU PQFP package only.

INPUT/OUTPUT PINS

Table 16.5 lists all the input/output pins, indicating their active level and when they are floated.

Table 16.5. Input/Output Pins

Name	Active Level	When Floated
D0–D31	HIGH	Bus Hold
DP0–DP3	HIGH	Bus Hold
A4–A31	HIGH	Bus, Address Hold

Table 16.6. Test Pins

Name	Input or Output	Sampled/ Driven On
TCK	Input	N/A
TDI	Input	Rising Edge of TCK
TDO	Output	Falling Edge of TCK
TMS	Input	Rising Edge of TCK

Table 16.7. Component and Revision ID (PGA)

i486 SX Microprocessor Stepping Name	Component ID	Revision ID
A0	04	20
B0	04	22

NOTE:
Table 16.7 shows the Component ID number and Revision ID number for the A-0 stepping of the Intel486 SX Microprocesor and Intel OverDrive Processor. When an Intel OverDrive Processor is installed in the system, the Component ID and Revision ID is provided by the Intel OverDrive Processor and not the Intel486 SX Microprocessor. The Component ID and Revision ID read by the BIOS/software may change when a Performance Upgrade Component, such as the Intel OverDrive Processor, is installed in an Intel486 SX Microprocessor based system.

16.6 Signal Description

With the exception of CLK2 and CLKSEL, all signals follow the same definition as the Intel486 microprocessor. The A.C. timing parameters for all of these signals are given in Tables 16.11–16.13.

CLOCK (CLK2)

CLK2 provides the fundamental timing for the Low Power Intel486 microprocessor. It is divided by two internally to generate the internal processor clock used for instruction execution. The internal clock is comprised of two phases, "phase one" and "phase two". Each CLK2 period is a phase of the internal clock. Figure 15.4 illustrates the relationship. If desired, the phase of the internal processor clock can be synchronized to a known phase by ensuring the pulse on the CLKSEL pin meets the applicable timings during cold boot (power-up reset).

All set-up, hold, float-delay and valid delay timings are referenced to the phase one of the clock.

The internal processor clock (CLK) is similar to the clock signal of the standard Intel486 microprocessor. All I/O signals get sampled on the rising edge of this signal, i.e. the rising edge of phase one. Thus it is important to synchronize the external circuitry with the phase one of CLK2.

CLKSEL

Clock Select pin selects the 2X mode required for the Low Power Intel486 CPU. This pin should be driven low after power-up and during the entire operation of the CPU. However, a well defined pulse is required on CLKSEL pin during cold boot (power-up reset) to establish the phase relationship of the 2X clock. The reset pulse width during cold reset should be at least 1 ms. As shown in Figure 16.5, the pulse on CLKSEL should be asserted by the end of reset (approximately 0.9 ms after driving reset active) and at least 30 CLK2 periods before the falling edge of reset.

Figure 16.6 shows the detailed timing definition of this pulse. The pulse on CLKSEL pin is only required during power-up reset. During all other times including warm resets the CLKSEL pin should be driven low and must be free of spikes or glitches. After the power-up reset, the system must track the phase of CLK2 at all times including during warm resets so that the input/output signals can be sampled at the appropriate clock edge. The phase relationship is described in the next section.

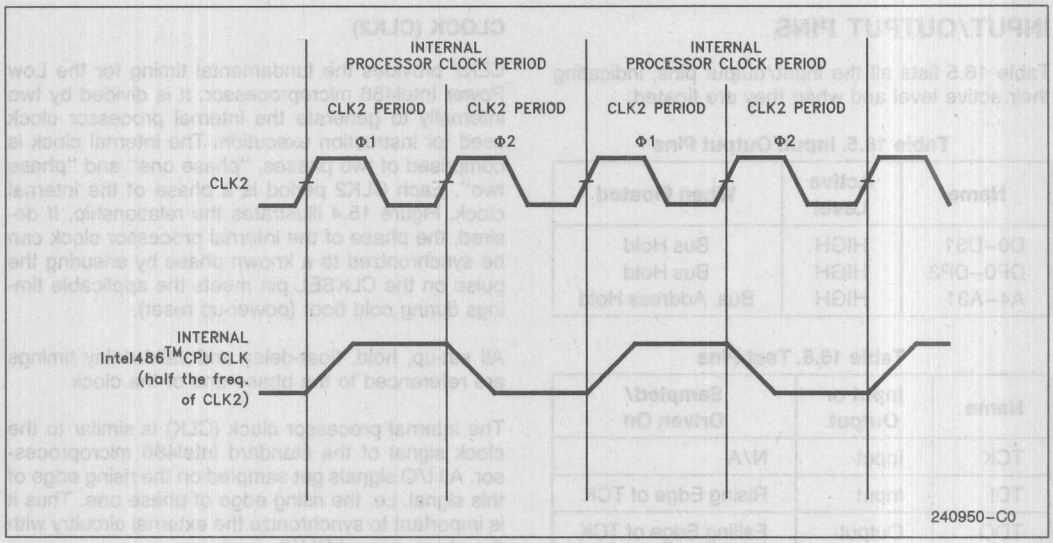
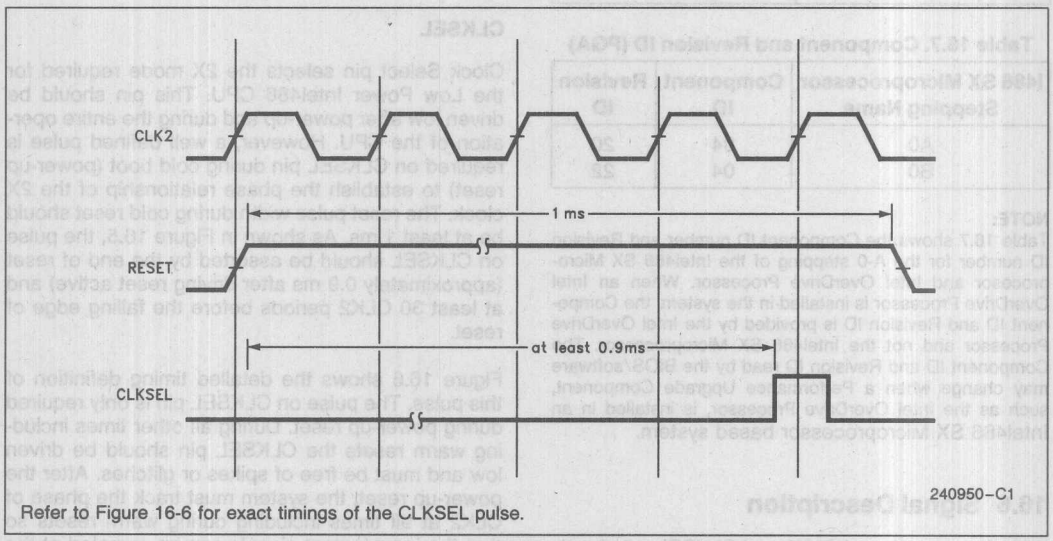


Figure 16.4. CLK2 Signal and Internal Processor Clock



Refer to Figure 16-6 for exact timings of the CLKSEL pulse.

Figure 16.5. CLKSEL Pulse with Reference to the Reset Pulse Width

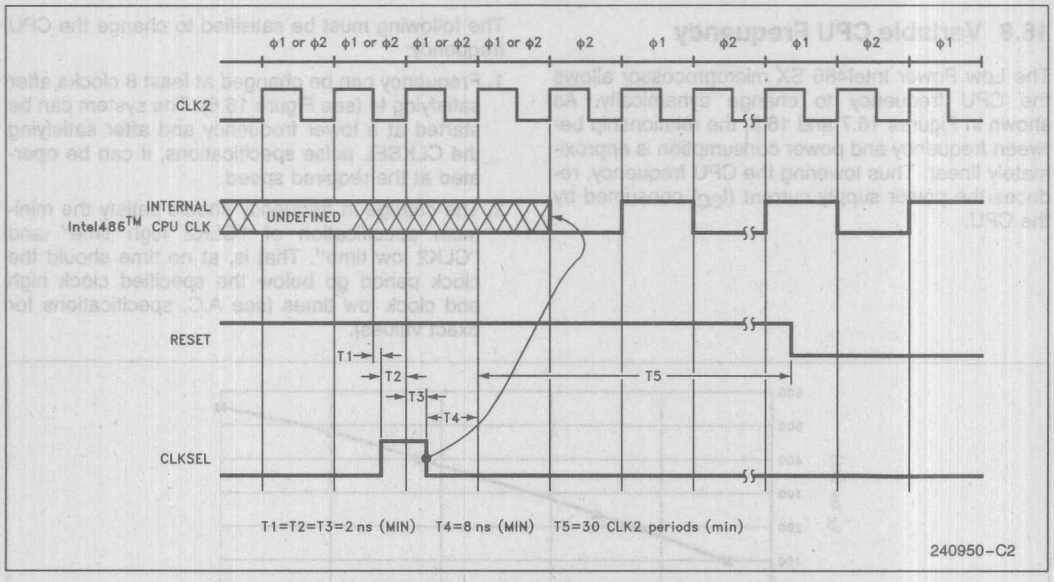


Figure 16.6. CLKSEL Timing Definition during Power-Up Reset

16.7 Architecture Overview

The Low Power Intel486 SX microprocessor is architecturally similar to the Intel486 SX CPU. Thus all bus cycles follow the same definition. The difference lies in the fact that the Low Power Intel486 SX CPU works with an external 2X clock input (CLK2). As shown in Figure 16.4, each of the internal processor clock (CLK) cycles is two CLK2 cycles wide. Thus a 25 MHz Low Power Intel486 microprocessor needs a 50 MHz clock input.

CLK2 provides the fundamental timing for the Low Power Intel486 SX CPU. It is divided by two internally to generate the internal processor clock (CLK) used for instruction execution. The internal clock is comprised of two phases, "phase one" and "phase two". Each CLK2 period is a phase of the internal clock. All Low Power Intel486 SX microprocessor inputs are sampled at the rising edge of phase 1. Each bus cycle is comprised of at least two bus states, T1 and T2. Each bus state in turn consists of two CLK2 cycles phase 1 and phase 2 of the bus state. The bus state diagram in Section 7.2.13 is valid for the Low Power Intel486 SX microprocessor.

NOTE:
The timing diagrams given for the Intel486 SX can be used for the Low Power Intel486 SX microprocessor. Read "CLK" signal as the internal clock of the CPU, with "CLK2" (the input clock of the Low Power Intel486 CPU) being twice the frequency of the internal processor clock as shown in Figure 16.4.

The following describes how the input signals are sampled and output signals are referenced with respect to the input clock (CLK2):

INPUT SIGNALS

The Low Power Intel486 SX CPU samples all its **synchronous** input signals (i.e., RDY#, BRDY#, BS8#, BS16#, KEN#, EADS#, BOFF#, HOLD and AHOLD) at the rising edge of phase 1, as long as proper setup and hold times relative to that clock edge are met.

The Low Power Intel486 SX CPU samples all its **asynchronous** input signals (i.e., RESET, INTR, NMI, A20M# FLUSH#, IGNNE#) at every other rising edge of the system clock (Phase 1), as long as proper setup and hold times relative to that clock edge are met.

OUTPUT SIGNALS

The A.C. timing specifications for output signals (i.e., valid and float delay timings) are specified with respect to the rising edge of the Phase 1 of the system clock. This holds true for all output signals including ADS# and PCHK#.

16.8 Variable CPU Frequency

The Low Power Intel486 SX microprocessor allows the CPU frequency to change dynamically. As shown in Figures 16.7 and 16.8, the relationship between frequency and power consumption is approximately linear. Thus lowering the CPU frequency, reduces the power supply current (I_{CC}) consumed by the CPU.

The following must be satisified to change the CPU frequency:

1. Frequency can be changed at least 8 clocks after satisfying t4 (see Figure 16.6). The system can be started at a lower frequency and after satisfying the CLKSEL pulse specifications, it can be operated at the required speed.
2. The change in frequency should satisfy the minimum specification of "CLK2 high time" and "CLK2 low time". That is, at no time should the clock period go below the specified clock high and clock low times (see A.C. specifications for exact values).

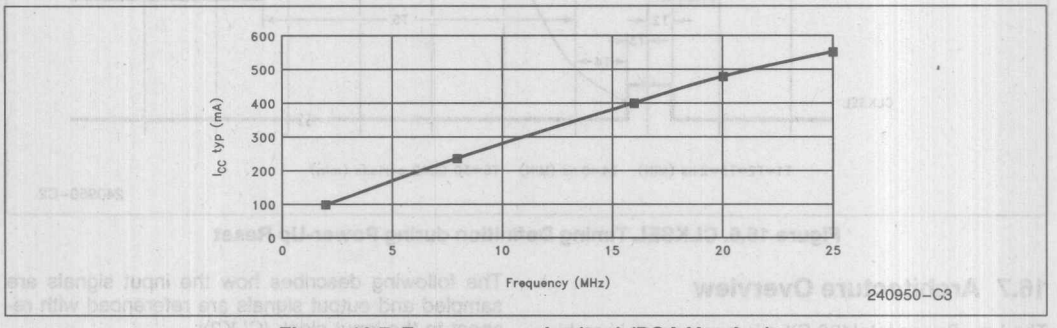


Figure 16.7. Frequency vs I_{CC}(typ) (PGA Version)

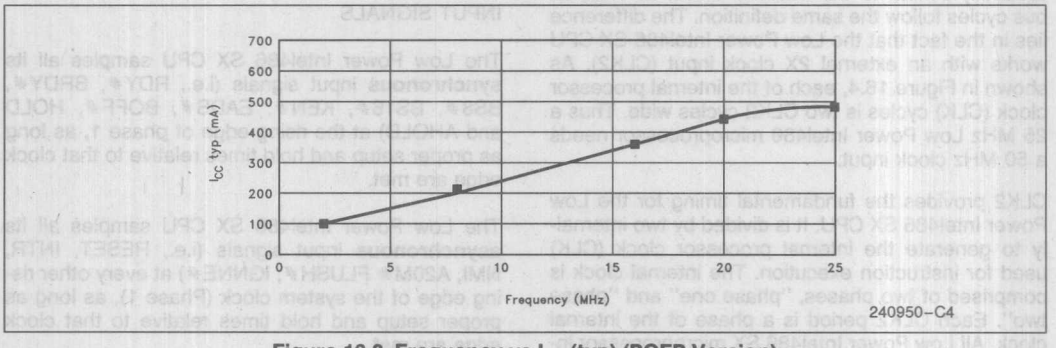


Figure 16.8. Frequency vs I_{CC}(typ) (PQFP Version)

16.9 D.C./A.C. Specifications

Table 16.6 provides the absolute maximum ratings. It is a stress rating only and functional operation at the maximums is not guaranteed. Functional operating conditions are given in Section 16.9.1 D.C. Specifications and 16.9.3 A.C. Specifications.

Table 16.6. Absolute Maximum Ratings

Case Temperature under Bias	−65°C to +110°C
Storage Temperature	−65°C to +150°C
Voltage on Any Pin with Respect to Ground	−0.5V to (V_{CC} +0.5V)
Supply Voltage with Respect to V_{SS}	−0.5V to +6.5V

16.9.1 D.C. SPECIFICATIONS

Table 16.7 provides the D.C. operating conditions for the Low Power Intel486 SX microprocessor (PGA Version) and the Intel487 SX Math CoProcessor installed in a low power system.

Functional Operating Range: V_{CC} = 5V ±10%; T_{case} = 0°C to +85°C.

Table 16.7. Low Power Intel486 SX Microprocessor D.C. Parametric Values (PGA Version)

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current CLK2 = 32 MHz = 40 MHz = 50 MHz		 525 600 700	 mA	(Note 3)
I_{CCF}	Power Supply Current with Intel486 SX CPU Tri-stated (floating) CLK2 = 32 MHz = 40 MHz = 50 MHz		 400 500 600	 mA	
I_{LI}	Input Leakage Current		±15	µA	(Note 4)
I_{IH}	Input Leakage Current		200	µA	(Note 5)
I_{IL}	Input Leakage Current		−400	µA	(Note 6)
I_{LO}	Output Leakage Current		±15	µA	
C_{IN}	Input Capacitance		20	pF	F_C = 1 MHz(7)
C_O	I/O or Output Capacitance		20	pF	F_C = 1 MHz(7)
C_{CLK}	CLK Capacitance		20	pF	F_C = 1 MHz(7)

NOTES:
1. This parameter is measured at:
   Address, Data BEn 4.0 mA
   Definition, Control 5.0 mA
2. This parameter is measured at:
   Address, Data BEn −1.0 mA
   Definition, Control −0.9 mA
3. Typical supply current
   I_{CC} = 400 @CLK2 = 32 MHz (Normal Operation)
   = 475 mA @CLK2 = 40 MHz
   = 500 mA @CLK2 = 50 MHz
   I_{CCF} = 325 mA @CLK2 = 32 MHz Intel486 CPU Tri-stated (Floating)
   = 400 mA @CLK2 = 40 MHz
   = 470 mA @CLK2 = 50 MHz
4. This parameter is for inputs without pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with pulldowns and V_{IH} = 2.4V.
6. This parameter is for inputs with pullups and V_{IL} = 0.45V.
7. Not 100% tested.

Table 16.8 provides the D.C. Operating Conditions for the Low Power Intel486 SX microprocessor (PQFP version) and the Intel487 SX Math CoProcessor installed in a low power system.

Functional Operating Range: V_{CC} = 5V − 10%, +5%; T_{case} = 0°C to +85°C

Table 16.8. Low Power Intel486™ SX Microprocessor D.C. Parametric Values (PQFP version)

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	24		V	(Note 2)
I_{CC}	Power Supply Current CLK2 = 32 MHz CLK2 = 40 MHz CLK2 = 50 MHz		 450 500 560	 mA	(Note 3)
I_{CCF}	Power Supply Current with Intel486 SX CPU in Power Down Mode		50	mA	(Note 7)
I_{LI}	Input Leakage Current		±15	μA	(Note 4)
I_{IH}	Input Leakage Current		200	μA	(Note 5)
I_{IL}	Input Leakage Current		−400	μA	(Note 6)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		10	pF	F_C = 1 MHz(7)
C_O	I/O or Output Capacitance		10	pF	F_C = 1 MHz(7)
C_{CLK}	CLK Capacitance		6	pF	F_C = 1 MHz(7)

* PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
1. This parameter is measured at:
   Address, Data, BEn 4.0 mA
   Definition, Control 5.0 mA
2. This parameter is measured at:
   Address, Data BEn −1.0 mA
   Definition, Control −0.9 mA
3. Typical supply current:
   I_{CC} 380 mA @ CLK2 = 32 MHz (Normal Operation)
   440 mA @ CLK2 = 40 MHz
   480 mA @ CLK2 = 50 MHz
4. This parameter is for inputs without pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with pulldowns and V_{IH} = 2.4V.
6. This parameter is for inputs with pullups and V_{IL} = 0.45V.
7. Not 100% tested.

16.9.2 POWER SUPPLY CURRENT vs FREQUENCY

Following is the power consumption of the Low Power Intel486 microprocessor or Intel487 SX Math CoProcessor installed in a low power system for different frequencies.

Table 16.9. Power Supply Current (I_{CC}) Values over Frequencies of Operation (PGA Version)

CLK2 Frequency (MHz)	Operating Frequency (MHz)	$I_{CC(max)}$ (mA)	$I_{CC(typ)}$ (mA)
4	2	150	100
16	8	325	235
32	16	525	400
40	20	600	475
50	25	700	550

Table 16.10. Power Supply Current (I_{CC}) Values over Frequencies of Operation (PQFP Version)

CLK2 Frequency (MHz)	Operating Frequency (MHz)	I_{CC} (max) (mA)	I_{CC} (typ) (mA)
4	2	150	100
16	8	250	210
32	16	450	380
40	20	500	440
50	25	560	480

16.9.3 A.C. SPECIFICATIONS

The following tables provide the A.C. specifications for the Low Power Intel486 SX microprocessors. They consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the rising edge of the phase 1 of the input system clock (CLK2), unless otherwise specified.

Table 16.11. Low Power Intel486™ SX—16 MHz Microprocessor/Intel487 SX Math CoProcessor A.C. Characteristics

V_{CC} = 5V ±10%; T_{case} = 0°C to +85°C; C_L = 50 pF(2) unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	2	16	MHz		Half of CLK2 Frequency
t_1	CLK2 Period	31	250	ns	16.9	
t_2	CLK2 High Time	10		ns	16.9	At 2V
t_3	CLK2 Low Time	10		ns	16.9	At 0.8V
t_4	CLK2 Fall Time		4	ns	16.9	2V to 0.8V
t_5	CLK2 Rise Time		4	ns	16.9	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, **FERR#***, BREQ, HLDA Valid Delay	3	26	ns	16.10	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		42	ns	16.10	After Clock Edge(1)

Microprocessor/Intel487™ SX Math CoProcessor A.C. Characteristics (Continued)

V_{CC} = 5V ±10%; T_{case} = 0°C to +85°C; C_L = 50 pF(2) unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_8	PCHK# Valid Delay	3	35	ns	16.10	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	35	ns	16.10	
t_9	BLAST#, PLOCK# Float Delay		42	ns	16.10	After Clock Edge(1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	30	ns	16.10	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		42	ns	16.10	After Clock Edge(1)
t_{12}	EADS# Setup Time	12		ns	16.11	
t_{13}	EADS# Hold Time	4		ns	16.11	
t_{14}	KEN#, BS16#, BS8# Setup Time	12		ns	16.11	
t_{15}	KEN#, BS16#, BS8# Hold Time	4		ns	16.11	
t_{16}	RDY#, BRDY# Setup Time	12		ns	16.11	
t_{17}	RDY#, BRDY# Hold Time	4		ns	16.11	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	12		ns	16.11	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	4		ns	16.11	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, **IGNNE#*** Setup Time	14		ns	16.11	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, **IGNNE#*** Hold Time	4		ns	16.11	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	10		ns	16.11	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	4		ns	16.11	
	CLKSEL	See Figures 16.5 and 16.6 for details on this signal. Figure 16.6 shows minimum timings required for the proper operation of the CPU. The pulse on CLKSEL can be of any length as long as the minimums are satisfied and the transitions from low to high occurs at the clock edge shown.				

*Present only in the Intel487 SX Math CoProcessor

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF.

Table 16.12. Low Power Intel486™ SX—20 MHz Microprocessor/Intel487™ SX Math CoProcessor A.C. Characteristics

V_{CC} = 5V ±10%; T_{case} = 0°C to +85°C; C_L = 50 pF(2) unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	2	20	MHz		Half of CLK2 Frequency
t_1	CLK2 Period	25	250	ns	16.9	
t_2	CLK2 High Time	8.5		ns	16.9	At 2V
t_3	CLK2 Low Time	8.5		ns	16.9	At 0.8V
t_4	CLK2 Fall Time		3	ns	16.9	2V to 0.8V
t_5	CLK2 Rise Time		3	ns	16.9	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, **FERR#***, BREQ, HLDA Valid Delay	3	23	ns	16.10	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		37	ns	16.10	After Clock Edge(1)
t_8	PCHK# Valid Delay	3	28	ns	16.10	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	28	ns	16.10	
t_9	BLAST#, PLOCK# Float Delay		37	ns	16.10	After Clock Edge(1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	26	ns	16.10	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		37	ns	16.10	After Clock Edge(1)
t_{12}	EADS# Setup Time	10		ns	16.11	
t_{13}	EADS# Hold Time	4		ns	16.11	
t_{14}	KEN#, BS16#, BS8# Setup Time	10		ns	16.11	
t_{15}	KEN#, BS16#, BS8# Hold Time	4		ns	16.11	
t_{16}	RDY#, BRDY# Setup Time	10		ns	16.11	
t_{17}	RDY#, BRDY# Hold Time	4		ns	16.11	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	12		ns	16.11	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	4		ns	16.11	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, **IGNNE#*** Setup Time	12		ns	16.11	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, **IGNNE#*** Hold Time	4		ns	16.11	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	6		ns	16.11	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	4		ns	16.11	

*Present only in the Intel487 SX Math CoProcessor

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF.

Table 16.13. Low Power Intel486™ SX—25 MHz Microprocessor/Intel487™ SX Math CoProcessor A.C. Characteristics

V_{CC} = 5V ±10%; T_{case} = 0°C to +85°C; C_L = 50 pF(2) unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	2	25	MHz		Half of CLK2 Frequency
t_1	CLK2 Period	20	250	ns	16.9	
t_2	CLK2 High Time	7		ns	16.9	At 2V
t_3	CLK2 Low Time	7		ns	16.9	At 0.8V
t_4	CLK2 Fall Time		2	ns	16.9	2V to 0.8V
t_5	CLK2 Rise Time		2	ns	16.9	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, **FERR#***, BREQ, HLDA Valid Delay	3	19	ns	16.10	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		28	ns	16.10	After Clock Edge(1)
t_8	PCHK# Valid Delay	3	24	ns	16.10	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	24	ns	16.10	
t_9	BLAST#, PLOCK# Float Delay		28	ns	16.10	After Clock Edge(1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	20	ns	16.10	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		28	ns	16.10	After Clock Edge(1)
t_{12}	EADS# Setup Time	9		ns	16.11	
t_{13}	EADS# Hold Time	4		ns	16.11	
t_{14}	KEN#, BS16#, BS8# Setup Time	9		ns	16.11	
t_{15}	KEN#, BS16#, BS8# Hold Time	4		ns	16.11	
t_{16}	RDY#, BRDY# Setup Time	9		ns	16.11	
t_{17}	RDY#, BRDY# Hold Time	4		ns	16.11	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	11		ns	16.11	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	4		ns	16.11	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, **IGNNE#*** Setup Time	11		ns	16.11	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, **IGNNE#*** Hold Time	4		ns	16.11	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	6		ns	16.11	

*Present only in the Intel487™ SX Math CoProcessor

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF.

Table 16.13. Low Power Intel486™ SX—25 MHz Microprocessor/Intel487™ SX Math CoProcessor A.C. Characteristics (Continued)

V_{CC} = 5V ±10%; T_{case} = 0°C to +85°C; C_L = 50 pF(2) unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	4		ns	2.3	
	CLKSEL	See Figures 16.5 and 16.6 for details on this signal. Figure 16.6 shows minimum timings required for the proper operation of the CPU. The pulse on CLKSEL can be of any length as long as the minimums are satisfied and the transitions from low to high occurs at the clock edge shown.				

*Present only in the Intel487™ SX Math CoProcessor

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF.

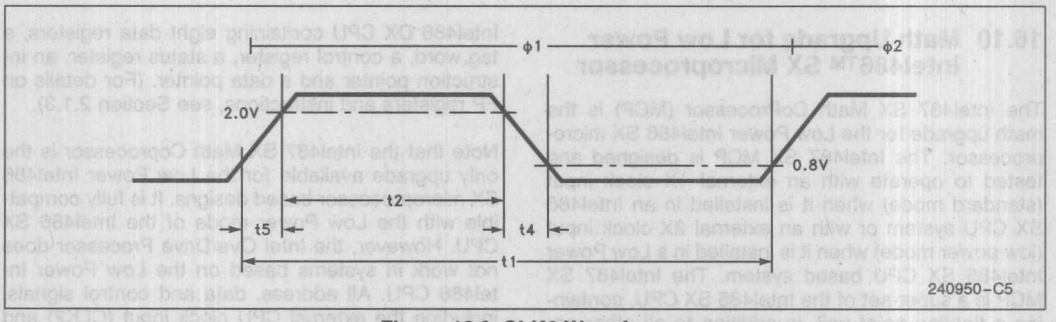


Figure 16.9. CLK2 Waveform

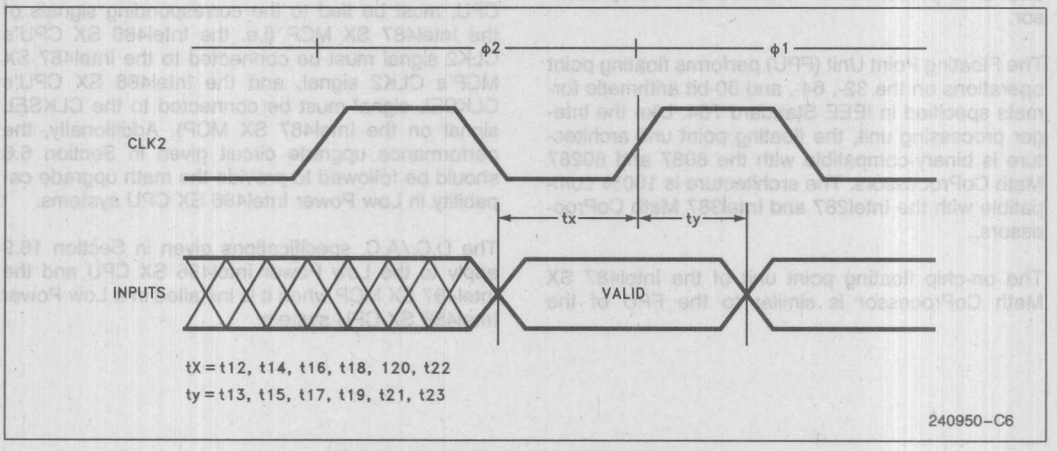


Figure 16.10. Setup and Hold Timings

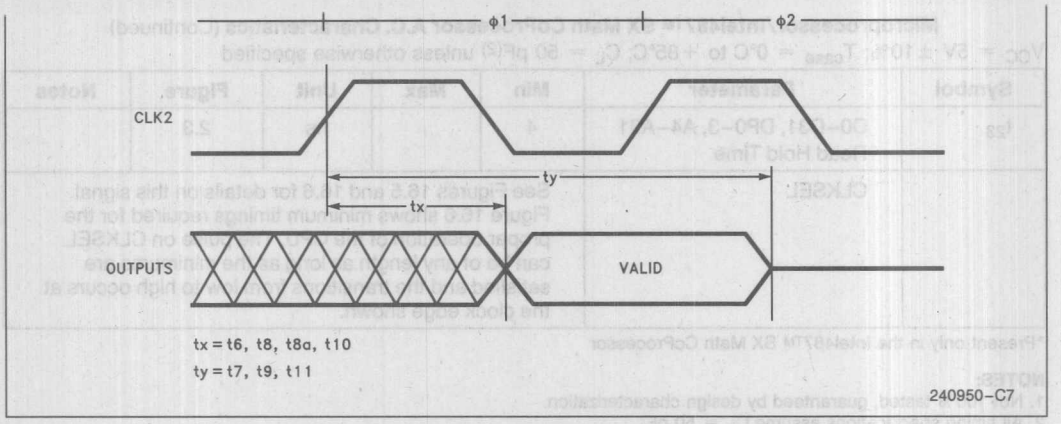


Figure 16.11. Valid and Float Delay Timings

16.10 Math Upgrade for Low Power Intel486™ SX Microprocessor

The Intel487 SX Math CoProcessor (MCP) is the math upgrade for the Low Power Intel486 SX microprocessor. The Intel487 SX MCP is designed and tested to operate with an external 1X clock input (standard mode) when it is installed in an Intel486 SX CPU system or with an external 2X clock input (low power mode) when it is installed in a Low Power Intel486 SX CPU based system. The Intel487 SX MCP is a super-set of the Intel486 SX CPU, containing a floating point unit, in addition to all other on-chip units present in the Intel486 SX microprocessor.

The Floating Point Unit (FPU) performs floating point operations on the 32-, 64-, and 80-bit arithmetic formats specified in IEEE Standard 754. Like the integer processing unit, the floating point unit architecture is binary-compatible with the 8087 and 80287 Math CoProcessors. The architecture is 100% compatible with the Intel287 and Intel387 Math CoProcessors.

The on-chip floating point unit of the Intel487 SX Math CoProcessor is similar to the FPU of the Intel486 DX CPU containing eight data registers, a tag word, a control register, a status register, an instruction pointer and a data pointer. (For details on FP registers and instructions, see Section 2.1.3).

Note that the Intel487 SX Math Coprocessor is the only upgrade available for the Low Power Intel486 SX microprocessor based designs. It is fully compatible with the Low Power mode of the Intel486 SX CPU. However, the Intel OverDrive Processor does not work in systems based on the Low Power Intel486 CPU. All address, data and control signals, including the external CPU clock input (CLK2) and the CLKSEL signal of the Low Power Intel486 SX CPU, must be tied to the corresponding signals of the Intel487 SX MCP (i.e. the Intel486 SX CPU's CLK2 signal must be connected to the Intel487 SX MCP's CLK2 signal, and the Intel486 SX CPU's CLKSEL signal must be connected to the CLKSEL signal on the Intel487 SX MCP). Additionally, the performance upgrade circuit given in Section 6.6 should be followed to provide the math upgrade capability in Low Power Intel486 SX CPU systems.

The D.C./A.C. specifications given in Section 16.9 apply to the Low Power Intel486 SX CPU and the Intel487 SX MCP when it is installed in a Low Power Intel486 SX CPU system.

16.10.1 PINOUT

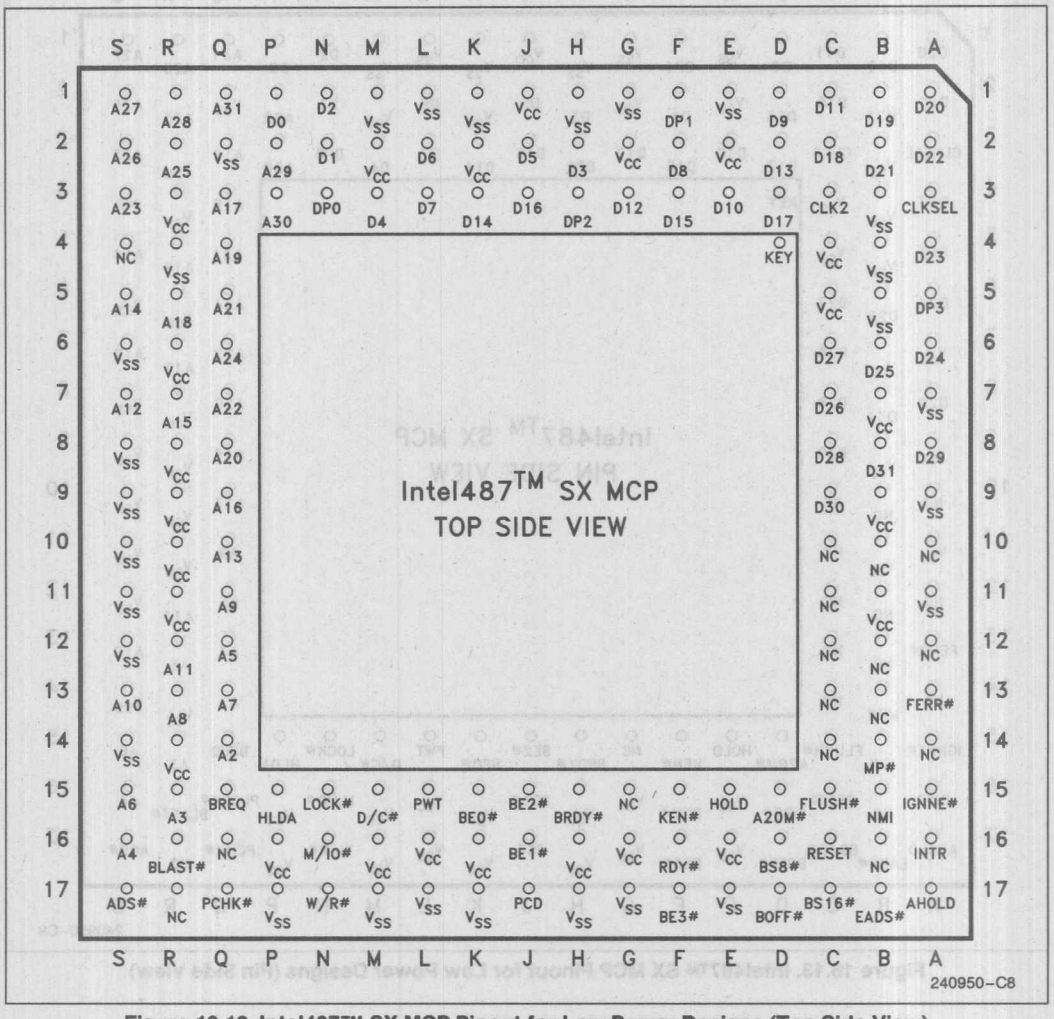


Figure 16.12. Intel487™ SX MCP Pinout for Low Power Designs (Top Side View)

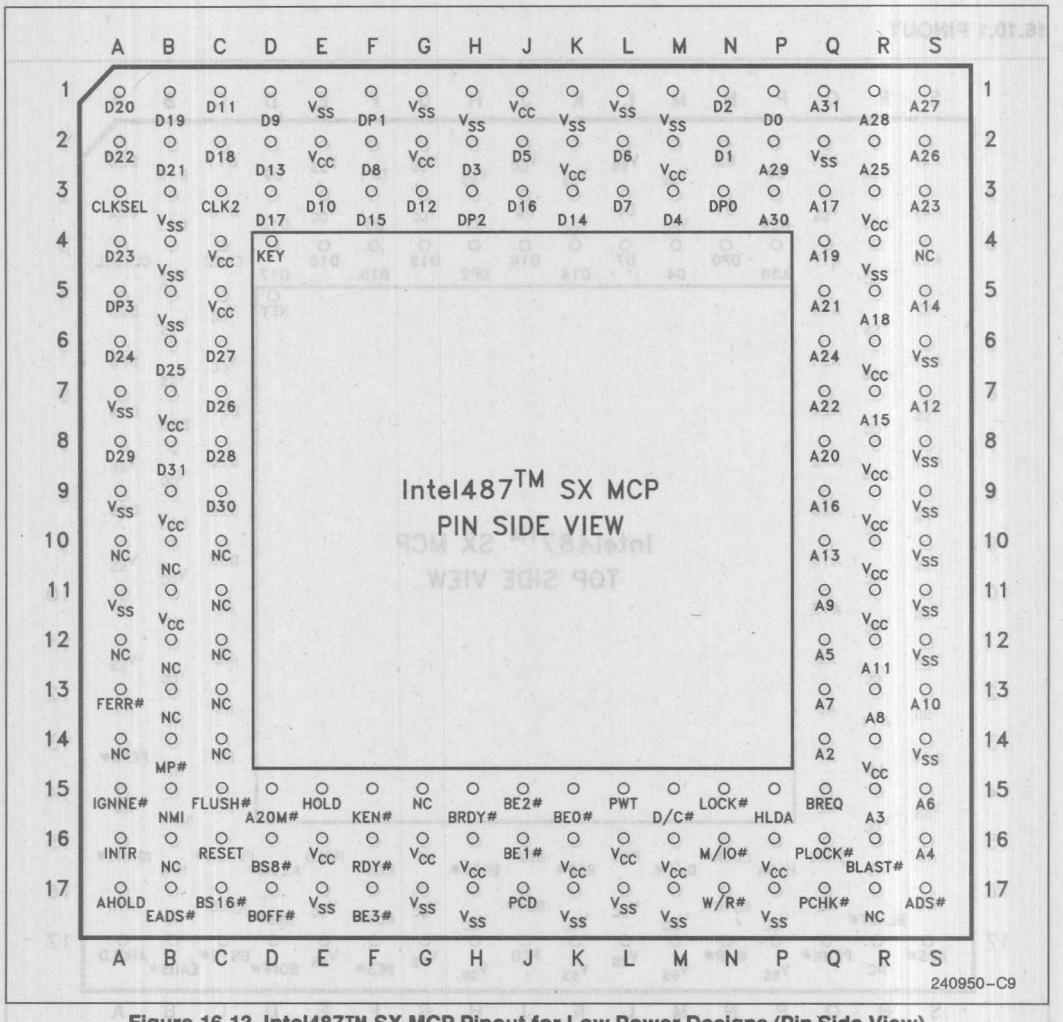


Figure 16.13. Intel487™ SX MCP Pinout for Low Power Designs (Pin Side View)

16.10.2 PIN REFERENCE OF Intel487™ SX Math CoProcessor

Table 16.14. Pin Cross Reference by Pin Name

Address		Data		Control		N/C	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	A10	B7	A7
A_3	R15	D_1	N2	ADS#	S17	A12	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	A14	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	B10	C4	B3
A_6	S15	D_4	M3	BE1#	J16	B12	C5	B4
A_7	Q13	D_5	J2	BE2#	J15	B13	E2	B5
A_8	R13	D_6	L2	BE3#	F17	B16	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16	C10	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17	C11	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15	C12	H16	G17
A_{12}	S7	D_{10}	E3	BREQ	Q15	C13	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16	C14	K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17	G15	K16	K1
A_{15}	R7	D_{13}	D2	CLK2	C3	R17	L16	K17
A_{16}	Q9	D_{14}	K3	CLKSEL	A3	S4	M2	L1
A_{17}	Q3	D_{15}	F3	D/C#	M15	D4	M16	L17
A_{18}	R5	D_{16}	J3	DP0	N3		P16	M1
A_{19}	Q4	D_{17}	D3	DP1	F1		R3	M17
A_{20}	Q8	D_{18}	C2	DP2	H3		R6	P17
A_{21}	Q5	D_{19}	B1	DP3	A5		R8	Q2
A_{22}	Q7	D_{20}	A1	EADS#	B17		R9	R4
A_{23}	S3	D_{21}	B2	FERR#	A13		R10	S6
A_{24}	Q6	D_{22}	A2	FLUSH#	C15		R11	S8
A_{25}	R2	D_{23}	A4	HLDA	P15		R14	S9
A_{26}	S2	D_{24}	A6	HOLD	E15			S10
A_{27}	S1	D_{25}	B6	IGNNE#	A15			S11
A_{28}	R1	D_{26}	C7	INTR	A16			S12
A_{29}	P2	D_{27}	C6	KEN#	F15			S14
A_{30}	P3	D_{28}	C8	LOCK#	N15			
A_{31}	Q1	D_{29}	A8	M/IO#	N16			
		D_{30}	C9	MP#	B14			
		D_{31}	B8	NMI	B15			
				PCD	J17			
				PCHK#	Q17			
				PWT	L15			
				PLOCK#	Q16			
				RDY#	F16			
				RESET	C16			
				W/R#	N17			

The Intel487 SX Math CoProcessor consists of all the Intel486 SX microprocessor signals with the following additional pins.

NUMERIC ERROR REPORTING		
FERR#	O	The **Floating point error** pin is driven active when a floating point error occurs. FERR# is similar to the ERROR# pin on the i387 Math CoProcessor. FERR# is included for compatibility with systems using DOS type floating point error reporting. FERR# is active LOW, and is not floated CONT1g bus hold.
IGNNE#	I	When the **ignore numeric error** pin is asserted, the Low Power Intel487 SX Math CoProcessor will ignore a numeric error and continue executing non-control floating point instructions. When IGNNE# is deasserted the Low Power Intel487 SX Math CoProcessor will freeze on a non-control floating point instruction, if a previous floating point instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set. IGNNE# is active LOW and is provided with a small internal pullup resistor. IGNNE# is asynchronous but setup and hold times t_{20} and t_{21} must be met to insure recognition on any specific clock.
Intel487™ SX Math CoProcessor INTERFACE		
MP#	O	The **math present** pin is used to signal the Intel486 SX microprocessor to float its outputs and get-off the bus. This pin can be used to check the presence of the Math CoProcessor in the two socket math upgrade circuit. It is active low and is never floated. MP# is driven low at power-up and remains active for the entire duration of the Intel487™ SX Math CoProcessor operation.
KEY PIN		
KEY		The **KEY** pin is an electrically non-functional pin which is used to insure the correct Intel487 SX Math CoProcessor orientation in a 169-pin socket. KEY pin is located at "D4" and is the 169th pin of the Intel487 SX MCP.

17.0 SUGGESTED SOURCES FOR Intel486™ SX MICROPROCESSOR/Intel OverDrive™ PROCESSOR ACCESSORIES

Following are some suggested sources of accessories for the Intel486 SX microprocessor/Intel OverDrive Processor. They are not an endorsement of any kind, nor a warranty of the performance of any of the listed products and/or companies.

Sockets

1. McKenzie Technology
   44370 Old Palmspring Blvd.
   Fremont, CA 94538
   Tel: (415) 651-2700
2. E-CAM Technology, Inc.
   14455 North Hayden Rd.
   Suite 208
   Scottsdale, AZ 85260
   Tel: (602) 443-1949
3. Augat Inc. (for sockets with decaps)
   Interconnection Products Group
   33 Perry Ave.
   P.O. Box 779
   Attleboro, MA 02703
   Tel: (508) 222-2202

OverDrive Processor Socket for the Intel OverDrive Processor

1. Amp Incorporated
   P.O. Box 3608
   Harrisburg, PA 17105-3608
   Tel: (800) 522-6752
2. Thomas & Betts
   200 Executive Center Dr.
   P.O. Box 24901
   Greenville, SC 29616-2401
   Tel: (201) 685-1600
3. Yamaichi Electronics Inc.
   1 420 Koll Circle Suite B
   San Jose, CA 95112
   Tel: (408) 452-0797

Heat Sinks/Fins

1. Thermalloy Inc.
   2021 West Valley View Lane
   Dallas, TX 75381-0839
   Tel: (214) 243-4321
2. E G & G Division
   60 Audubon Road
   Wakefield, MA 01880
   Tel: (617) 245-5900

TTL Crystals/Oscillators

1. NEL Frequency Controls, Inc.
   357 Beloit Street
   Burlington, WI 53105
   Tel: (414) 763-3591
2. M-Tron
   P.O. Box 630
   Yankton, SD 57078
   Tel: (605) 665-9321

Debugging Tower

1. Emulation Technology
   2344 Walsh Ave., Building F
   Santa Clara, CA 95051
   Tel: (408) 982-0664

18.0 REVISION HISTORY

Revision -004 of the Intel486 SX Microprocessor Data Book contains many updates and improvements to the original version. A revision summary of major changes is listed below:

The sections significantly revised since version -001 are:

Cover Page	Added Intel486 SX CPU PQFP package information.
Figure 1.1a	Corrected the Intel486 SX CPU pinout. Pin D6 at location F2 has been changed to pin D8.
Figure 1.1b	Corrected the Intel487 SX MCP pinout. Pin D6 at location F2 has been changed to pin D8.
Figure 1.2a	Corrected the Intel486 SX CPU pinout. Pin D6 at location F2 has been changed to pin D8.
Figure 1.2b	Corrected the Intel487 SX MCP pinout. Pin D6 at location F2 has been changed to pin D8.
Figure 1.3	Added Intel486 SX CPU PQFP pinout.
Table 1.1c & Table 1.1d	Added Intel486 SX CPU PQFP package Pin Cross Reference table.
Quick Pin Reference	The PWT and PCD functional description has been clarified. New signals for only the Intel486 SX CPU PQFP package have been added.
Table 1.2	The MP# pin has been added to the input pin list for the Intel487 SX MCP.
Table 1.3	The UP# pin has been added to the input pin list for the Intel486 SX CPU PQFP package.
Table 1.5	Added a new pin table for the Intel486 SX CPU PQFP package test pins.
Component & Revision ID	Added note about Component ID and Revision ID numbers for the Intel486 SX CPU and Intel487 SX MCP.
Section 2.0	A description of the features of the Intel486 SX CPU in the PQFP package has been added.
Section 6.2.9	Added description of HOLD recognition during BOFF#.
Section 6.2.12	Added PCD and PWT description when paging disabled.
Section 6.2.15	Corrected explanation of A20M# sampling during reset.
Section 6.2.16	Added section on Performance Upgrade support signal description for the Intel486 SX CPU PQFP package.
Section 6.2.17	Added section on Boundary Scan Test Signals for the Intel486 SX CPU PQFP package.
Figure 6.4	Added additional details on signal sampling during RESET.
Figure 6.5.1	Added details about FERR# signal.
Figure 6.6	Corrected Intel487 SX MCP signal name on the Performance Upgrade circuit. The HOLD signal from the Intel487 SX MCP should be connected to the HOLD signal from the Intel486 SX CPU PGA package; the DATA signal from the Intel487 SX MCP should NOT be connected to the HOLD signal from the Intel486 SX CPU PGA package.
Figure 6.7	Added a Performance Upgrade circuit for the Intel486 SX CPU in the PQFP package.
Figure 7.30	Added HOLD to state transition between Tb and T1b.
Section 8.1	Corrected description of how BIST is initiated.
Section 8.2.2	Added information about moves to TR4 and TR5.
Section 8.4	Corrected description of how the Tri-State Output Test Mode is initiated.
Section 8.5	Section 8.5 describes the Boundary Scan feature of the PQFP version of the Intel486 SX CPU.
Table 13.2	16 MHz and 25 MHz D.C. specifications were added for the Intel486 SX CPU and for the PGA package.
Table 13.3	Table 12.3 is a new table which shows the 16 MHz, 20 MHz, and 25 MHz D.C. specifications for the Intel486 SX CPU for the PQFP package. The maximum and typical I_{CC} specification for the 20 MHz Intel487 SX MCP has been reduced.
Table 13.4	16 MHz and 25 MHz D.C. specifications were added for the Intel487 SX MCP.
Table 13.5	Added table with 16 MHz A.C. specifications for the Intel486 SX CPU and Intel487 SX MCP.

Table 13.6	Added improved 20 MHz A.C. specifications for T6 through T11 for the Intel486 SX CPU and Intel487 SX MCP. Minimum frequency was changed from 16 MHz to 8 MHz.
Table 13.7	Added table with 25 MHz A.C. specifications for the Intel486 SX CPU and Intel487 SX MCP.
Section 13.4.2	Added Derating curves for Intel486 SX CPU in PQFP.
Figure 14.3a & Figure 14.3b & Figure 14.3c & Table 14.2	Figures 13.3a, 13.3b, 13.3c and Table 13.2 were added to show the specifications for the 196 pin PQFP package.
Table 14.4	Table 13.4 was added to provide the θ_{JA} and θ_{JC} values for the 196 pin PQFP package.
Table 14.5 & Table 14.6	Corrected maximum T_A tables for the Intel486 SX CPU for the PGA package and the Intel487 SX MCP.
Table 14.7	Table 13.7 was added to show the maximum T_A values for the Intel486 SX CPU for the 196 pin PQFP package.
Section 15.0	Corrected address for AMP Incorporated.
Appendix B	A FINIT instruction was added to the "restore_EFLAGS" section of the Intel Recommended CPU Identification Code.

Significant revisions since -002 are listed below:

Intel OverDrive Processor information/specifications have been added throughout the document, including 33 MHz specifications for the OverDrive Processor. Section 8.0 contains OverDrive Processor specific information.

A new section was added containing a summary of differences between the Intel486 SX Microprocessor in PGA and PQFP packages.

Intel OverDrive Processor information/specifications have been added throughout the document. Section 13.0 contains OverDrive Processor specific information.

33 MHz Intel486 SX Microprocessor specifications have been added throughout the document.

Low Power Intel486 DX CPU information/specifications have been added. Section 16.0 contains Low Power specific information.

Cover Page	Listed boundary scan compatibility feature for PQFP package.
Pinout	Noted the pinout and pin description for the Intel487 SX MCP are identical to the Intel OverDrive Processor except for the MP# pin is redefined as the UP# pin on the Intel OverDrive Processor.
Pin Description	More clearly defined A20M# pin.
Table 1.1d	Corrected Bus Request instruction from BREQ# to BREQ.
Table 1.6	Added B0 stepping information for Intel486 SX microprocessor along with OverDrive Processor stepping information.
Table 6.3	Added stepping info for Intel487 SX MCP and OverDrive Processor.
Section 8.1	Added number of clocks required for BIST at 25 MHz.
Section 8.5.5	Removed FERR# as a control pin for MISCCTL.
Chapter 12	New chapter on the differences with the Intel486 SX Microprocessor in PGA vs PQFP.
Chapter 13	New chapter on OverDrive Processor.
Section 14.3	Added D.C. specs for Intel 33 MHz 486 SX microprocessor.
Table 13.5	Changed Hold times for Intel 16 MHz 486 SX microprocessor 4 ns to 3 ns.
Section 13.4	Added A.C. specs for Intel486™ SX 33 MHz for PGA and PQFP packages.
Section 13.4	Added Boundary Scan Test signals and timing diagrams.
Chapter 16	New section for Low Power Intel486 SX microprocessor.

The sections significantly revised since version -003 are:

Section 6.5	Added clarification for the built in self test (BIST) during reset.
Section 7.2.9	Added explanation of bus hold and hold acknowledge protocol.
Figure 7.26b	Added figure to illustrate HOLD request acknowledge during BOFF#.
Table 14.2	Changed PGA I_{CC} Max specifications for 16 MHz and 33 MHz.
Table 14.3	Changed PQFP I_{CC} Max specifications for all frequencies.
Table 14.8	Changed Max specification of t_9 BLAST#, PLOCK# Float Delay.

Table 14.9	Changed Min and Max specifications for Boundary Scan Test Inputs.
Table 15.4	Changed maximum ambient temperature specifications for 16 MHz and 33 MHz CPUs in PGA package.
Table 15.7 and Table 15.8	Changed maximum ambient temperature specifications for all frequencies of CPUs in PQFP package.

APPENDIX A
OverDrive™ PROCESSOR INSTALLATION MADE EASY

OEM PC manufacturers can make OverDrive Processor installation easier for the end user and reseller. Simple and foolproof OverDrive Processor installation results in fewer broken PC boards and less end user and reseller frustration. The following is a brief list of ways to make OverDrive Processor installation simple and foolproof for the end user and reseller:

1. Intel has packaged the Intel487 SX Math CoProcessor in a 169 pin PGA package. The 169th pin insures that the Intel487 SX Math CoProcessor fits into a 169 pin socket in only the correct orientation. Supplying a 169 pin socket as the OverDrive Processor eliminates the possibility of damaging the PC board due to end users and resellers powering up the system with the OverDrive Processor in an incorrect orientation(1).
2. Because of the high pin count of the Intel487 SX Math CoProcessor package, the insertion force required for a standard PGA socket is excessive. Even most Low Insertion Force sockets require approximately 30 lbs. (13.5 kg) of insertion force. A Zero Insertion Force socket insures that the OverDrive Processor installation insertion force does not damage the PC board. Be sure to allow enough clearance for the socket handle when a Zero Insertion Force socket is used.
3. Make the OverDrive Processor socket easily accessible to the end user. (i.e., Do not place the OverDrive Processor socket under a disk drive.)
4. Label the Intel487 SX Math CoProcessor socket and the location of pin 1 by silk screening this information on the PC board. See Figure A.
5. Describe the Intel487 SX Math CoProcessor installation procedure in the PC's User's Manual.

NOTE:
1. Many socket manufacturers use a standard grid array for PGA sockets. The socket cover contains more than the required number of pin holes, but contacts are only loaded where required for electrical connection. For the 169th pin to assist the end user and reseller in installing the chip with the correct orientation, it is essential that the socket have only 169 pin holes in the cover. The only sure way to ascertain that a socket has only 169 pin holes is to order samples and do a visual check.

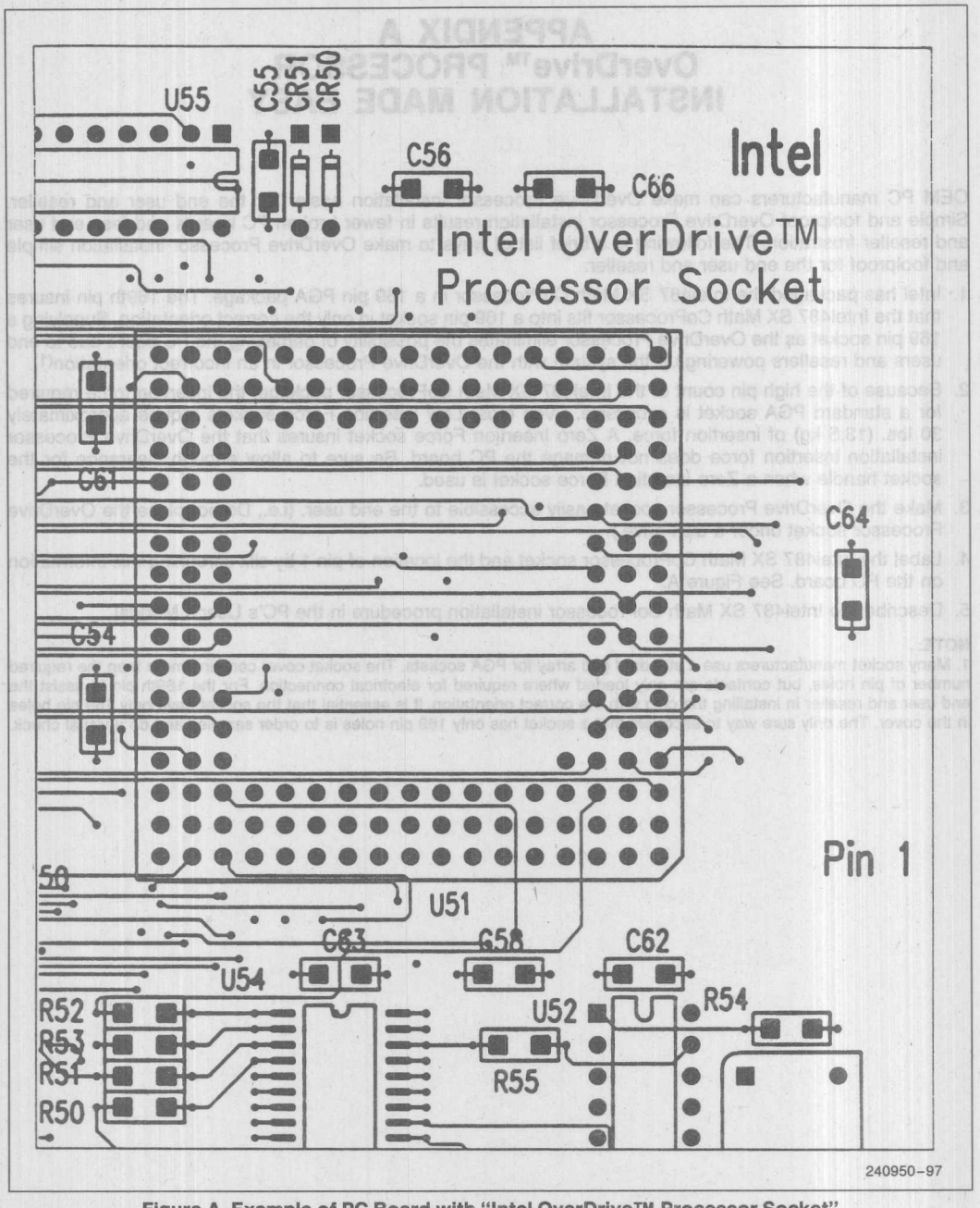


Figure A. Example of PC Board with "Intel OverDrive™ Processor Socket" and "Pin 1" Silk Screened onto PC Board

APPENDIX B
INTEL RECOMMENDED CPU IDENTIFICATION CODE

The CPU identification assembly code will determine for the user which Intel microprocessor and if a Intel Math CoProcessor is installed in the system. If a 486 microprocessor has been installed, the program will determine if the CPU is with/without a floating point unit. This code should be executed so the system can be configured for a particular application, which may depend on the microprocessor and Math CoProcessor installed in the system.

```
        TITLE CPUID
        DOSSEG
        .model  small

        .stack  100h

        .data
fp_status       dw      ?
id_mess         db      "This system has a$"
fp_8087         db      "and an 8087 Math CoProcessor$"
fp_80287        db      "and an 287 Math CoProcessor$"
fp_80387        db      "and an 387 Math CoProcessor$"
c8086           db      "n8086/8088 microprocessor$"
c286            db      "n80286 microprocessor$"
c386            db      "386 microprocessor$"
c486            db      "486 DX microprocesser/487 SX Math
                        "CoProcessor$"
c486nfp         db      "486 SX Microprocesser$"
period          db      ".$",13,10
present_86      dw      0
present_286     dw      0
present_386     dw      0
present_486     dw      0
;
;       The purpose of this code is to allow the user the ability to identify
;       the processor and coprocesor that is currently in the system. The
;       algorithm of the program is to first determine the processor id.
;       When that is accomplished, the program continues to then identify
;       whether a coprocessor exists in the system. If a coprocessor or
;       integrated coprocessor exists, the program will identify the
;       coprocessor id. If one does not exist, the program then terminates.
;

        .code
start:
                mov     ax,@data
                mov     ds,ax                   ; set segment register

                mov     dx,offset id_mess       ;print header message
                mov     ah,9h
                int     21h
```

```
;
;       8086 check
;       Bits 12-15 are always set on the 8086 processor.
;
                pushf                           ; save EFLAGS
                pop     bx                      ; store EFLAGS in BX
                mov     ax,0fffh                ; clear bits 12-15
                and     ax,bx                   ;       in EFLAGS
                push    ax                      ; store new EFLAGS value on stack
                popf                            ; replace current EFLAGS value
                pushf                           ; set new EFLAGS
                pop     ax                      ; store new EFLAGS in AX
                and     ax,0f000h               ; if bits 12-15 are set, then CPU
                cmp     ax,0f000h               ;       is an 8086/8088
                mov     dx,offset c8086         ; store 8086/8088 message
                mov     present_86,1            ; turn on 8086/8088 flag
                je      check_fpu               ; if CPU is 8086/8088, check for
                                                ; 8087

;
;       80286 CPU Check
;       Bits 12-15 are always clear on the 80286 processor.
;

                or      bx,0f000h               ; try to set bits 12-15
                push    bx
                popf
                pushf
                pop     ax
                and     ax,0f000h               ; if bits 12-15 are cleared, then
                mov     dx,offset c286          ;       CPU is an 80286
                mov     present_86,0            ; turn off 8086/8088 flag
                mov     present_286,1           ; turn on 80286 flag
                jz      check_fpu               ; if CPU is 80286, check for 80287

;
;       386 CPU check
;       The AC bit, bit #18, is a new bit introduced in the EFLAGS register
;       on the 486 DX CPU to generate alignment faults. This bit can be set
;       on the 486 DX CPU, but not on the 386 CPU.
;

                mov     bx,sp                   ; save current stack pointer to
                                                ; align it
                and     sp,not 3                ; align stack to avoid AC fault
                db      66h
                pushf                           ; push original EFLAGS
                db      66h
                pop     ax                      ; get original EFLAGS
                db      66h
                mov     cx,ax                   ; save original EFLAGS
                db      66h                     ; xor EAX,40000h
                xor     ax,0                    ; flip AC bit in EFLAGS
                dw      4                       ; upper 16-bits of xor constant
                db      66h
                push    ax                      ; save for EFLAGS
                db      66h
                popf                            ; copy to EFLAGS
```

```
                db      66h
                pushf                   ; push EFLAGS
                db      66h
                pop     ax              ; get new EFLAGS value
                db      66h
                xor     ax,cx           ; if AC bit cannot be changed,
                                        ; CPU is
                mov     dx,offset c386  ; store 386 message
                mov     present_286,0   ; turn off 80286 flag
                mov     present_386,1   ; turn on 386 flag
                je      check_fpu       ; if CPU is 386, now check for
                                        ;  80287/80387
;
;       486 DX CPU and 486 DX CPU w/o FPU checking
;
                mov     dx,offset c486nfp   ; store 486NFP message
                mov     present_386,0   ; turn off 386 flag
                mov     present_486,1   ; turn on 486 flag
;
;       Co-processor checking begins here for the 8086/80286/386 CPUs.
;       The algorithm is to determine whether or not the floating-point
;       status and control words can be written to, the correct coprocessor
;       is then determined depending on the processer id. Coprocessor checks
;       are first performed for an 8086, 80286 and a 486 DX CPU. If the
;       coprocessor id is still undetermined, the system must contain a 386
;       CPU. The 386 CPU may work with either an 80287 or an 80387. The
;       infinity of the coprocessor must be checked to determine the correct
;       coprocessor id.
;
check_fpu:                              ; check for 8087/80287/80387
                fninit                  ; reset FP status word
                mov     fp_status,5a5ah ; initialize temp word to non-zero
                                        ; value
                fnstsw  fp_status       ; save FP status word
                mov     ax,fp_status    ; check FP status word
                cmp     al,0            ; see if correct status with
                                        ; written
                jne     print_one       ; jump if not Valid, no NPX
                                        ; installed
                fnstcw  fp_status       ; save FP control word
                mov     ax,fp_status    ; check FP control word
                and     ax,103fh        ; see if selected parts looks OK
                cmp     ax,3fh          ; check that ones and zeroes
                                        ; correctly read
                jne     print_one       ; jump if not Valid, no NPX
                                        ; installed

                cmp     present_486,1   ; check if 486 flag is on
                je      is_486          ; if so, jump to print 486 message
                jmp     not_486         ; else continue with 386 checking
is_486:
                mov     dx,offset c486  ; store 486 message
                jmp     print_one
```

```
not_486:
                cmp     present_386,1           ; check if 386 flag is on
                jne     print_87_287            ; if 386 flag not on, check NPX for
                                                ;  8086/8088/80286
                mov     ah,9h                   ; print out 386 CPU ID first
                int     21h
;
;       80287/80387 check for the 386 CPU
;
                fldl                            ; must use default control from
                                                ; FNINIT
                fldz                            ; form infinity
                fdiv                            ; 8087/80287 says +inf = inf
                fld     st                      ; form negative infinity
                fchs                            ; 80387 says +inf <> -inf
                fcompp                          ; see if they are the same and
                                                ; remove them
                fstsw   fp_status               ; look at status from FCOMPP
                mov     ax,fp_status
                mov     dx,offset fp_80287      ; store 80287 message
                sahf                            ; see if infinities matched
                jz      restore_EFLAGS          ; jump if 8087/80287 is present
                mov     dx,offset fp_80387      ; store 80387 message
restore_EFLAGS:
                finit                           ; clear any pending fp exception
                mov     ah,9h                   ; print NPX message
                int     21h
                db      66h
                push    cx                      ; push ECX
                db      66h
                popf                            ; restore original EFLAGS register
                mov     sp,bx                   ; restore original stack pointer
                jmp     exit
print_one:
                mov     ah,9h                   ; print out CPU ID with no NPX
                int     21h
                jmp     exit
print_87_287:
                mov     ah,9h                   ; print out 8086/8088/80286 first
                int     21h
                cmp     present_86,1            ; if 8086/8088 flag is on
                mov     dx,offset fp_8087       ; store 8087 message
                je      print_fpu
                mov     dx,offset fp_80287      ; else CPU = 80286, store 80287
                                                ; message
print_fpu:
                mov     ah,9h                   ; print out NPX
                int     21h
                jmp     exit
exit:
                mov     dx,offset period        ; print out a period of end message
                mov     ah,9h
                int     21h

                mov     ax,4c00h                ; terminate program
                int     21h

                end     start
```

APPENDIX C
3.3V Intel486™ SX MICROPROCESSOR

- **Lower Supply Voltage Required**
  - **— Operates from 3.0V to 3.6V**
  - **— 50% Lower Power than 5V CPU**
  - **— Lower Power Consumption Produces Less Heat**
  - **— Longer Battery Life for Portable Applications**
- **Binary Compatible with Large Software Base**
  - **— MS-DOS*, OS/2**, Windows***
  - **— UNIX*** System V/386**
  - **— iRMX®, iRMK Kernels**
- **High Integration Enables On-Chip**
  - **— 8 Kbyte Code and Data Cache**
  - **— Paged, Virtual Memory Management**
- **Easy To Use**
  - **— Built in Self Test**
  - **— Intel Software Support**
  - **— Extensive Third Party Software Support**
- **196-Lead PQFP Package**
- **High Performance Design**
  - **— Intel486 One Clock Instruction Core**
  - **— 25 MHz, 20 MHz, and 16 MHz Clock Frequencies**
  - **— 80 Mbytes/sec Burst Bus at 25 MHz**
  - **— CHMOS V Process Technology**
  - **— Dynamic Bus Sizing for 8-, 16-, 32-Bit Busses**
- **Complete 32-Bit Architecture**
  - **— Address and Data Busses**
  - **— Registers**
  - **— 8-, 16-, 32-Bit Data Types**
- **IEEE 1149.1 Boundary Scan Compatibility**
  - **— For Surface Mount Production Testing**

2

The 3.3V Intel486 SX microprocessor provides a low-cost entry point to powerful Intel486 microprocessor based portable computing. The Intel486 SX microprocessor is a binary compatible derivative of the Intel486 DX microprocessor, giving it access to the $50 billion installed software base of over 50,000 MS-DOS, Windows, OS/2, and UNIX system V/386 applications. The Intel486 SX microprocessor has the same integrated RISC integer core, 8 Kbyte cache memory, and memory management unit as the Intel486 DX microprocessor.

The high-performance RISC integer core of the Intel486 SX microprocessor executes frequently-used instructions in one clock cycle. An 8 Kbyte unified code and data cache allow this performance level to be sustained. An 80 Mbyte/sec burst bus at 25 MHz ensures high system throughput even with inexpensive DRAMs. The 25 MHz Intel486 SX microprocessor has a Norton SI V5.0 rating of 54.1, a Dhrystone MIPS rating of 20.1 and a 13.3 SPEC Integer rating. It provides up to 70% greater performance than a 33 MHz Intel386™ DX microprocessor with an external cache (depending on the application) at a lower system cost.

This documentation lists the A.C. and D.C. specifications of the 3.3V Intel486 SX microprocessor and is a supplement to the Intel486™ SX Microprocessor/Intel487™ SX Math Coprocessor Data Book (Order Number 240950-002).

i386, Intel386, i387, Intel387, i486, Intel486, i487, Intel487 are trademarks of Intel Corp.
*MS-DOS and Windows are trademarks of Microsoft Corp.
**OS/2 is a trademark of IBM.
***UNIX is a registered trademark of UNIX Systems Labs.

Maximum Ratings

The ratings listed in this section are stress ratings only, and functional operation at the maximums is not guaranteed. Functional operating conditions are given in the D.C. Specifications and A.C. Specifications sections.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the Intel486 SX microprocessor contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

Absolute Maximum Ratings

Case Temperature under Bias	−65°C to +110°C
Storage Temperature	−65°C to +150°C
Voltage on Any Pin with Respect to Ground	−0.5V to V_{CC} + 0.5V
Supply Voltage with Respect to V_{SS}	−0.5V to +6.5V

D.C. SPECIFICATIONS

Functional operating range: V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C

3.3V Intel486™ SX Microprocessor D.C. Parametric Values

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input LOW Voltage	−0.3	+0.8	V	
V_{IH}	Input HIGH Voltage	2.0	V_{CC} + 0.3	V	
V_{OL}	Output LOW Voltage I_{OL} = 4/5 mA I_{OL} = 0.1 mA		* 0.45 0.2	 V V	(Note 1)
V_{OH}	Output HIGH Voltage I_{OH} = −1/0.9 mA I_{OH} = −0.1 mA	 2.4 V_{CC} − 0.2		 V V	(Note 2)
I_{CC}	Supply Current CLK = 8 MHz CLK = 16 MHz CLK = 20 MHz CLK = 25 MHz		 150 260 290 340	 mA mA mA mA	(Note 3)
I_{LI}	Input Leakage Current		±15	µA	(Note 4)
I_{IH}	Input Leakage Current		200	µA	(Note 5)
I_{IL}	Input Leakage Current		−400	µA	(Note 6)
I_{LO}	Output Leakage Current		±15	µA	
C_{IN}	Input Capacitance		10	pF	F_C = 1 MHz(7)
C_{OUT}	Output or I/O Capacitance		10	pF	F_C = 1 MHz(7)
C_{CLK}	CLK Capacitance		6	pF	F_C = 1 MHz(7)

* PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:

1. This parameter is measured at:
   Address, Data, BEn 4.0 mA
   Definition, Control 5.0 mA
2. This parameter is measured at:
   Address, Data, BEn −1.0 mA
   Definition, Control −0.9 mA
3. Typical supply current (3.3V):
   ICC @ 8 MHz = 90 mA
   @ 16 MHz = 180 mA
   @ 20 MHz = 210 mA
   @ 25 MHz = 250 mA
4. This parameter is for inputs without pull ups or pull downs and $0V \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with pull downs and V_{IH} = 2.4V.
6. This parameter is for inputs with pull ups and V_{IL} = 0.45V.
7. Not 100% tested.

A.C. SPECIFICATIONS

The A.C. Specifications given in the following tables consist of output delays, input setup requirements and input hold requirementts. All A.C. specifications are relative to the rising edge of the CLK signal.

A.C. specifications measurement is defined by the timing diagrams in the base Intel486 SX documentation (refer to the cover page of this document for the order number). The Boundary Scan (JTAG) timing waveforms are included in this document following the A.C. Specifications section.

16 MHz 3.3V Intel486™ SX Microprocessor A.C. Characteristics

V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Figure	Units	Notes
	Frequency	8	16		MHz	1X Clock
t_1	CLK Period	62.5	125	12.1	ns	
t_{1a}	CLK Period Stability		0.1%	12.1	ns	Adjacent Clocks
t_2	CLK High Time	20		12.1	ns	at 2V
t_3	CLK Low Time	20		12.1	ns	at 0.8V
t_4	CLK Fall Time		8	12.1	ns	2V to 0.8V
t_5	CLK Rise Time		8	12.1	ns	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Valid Delay	3	26	12.5	ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		42	12.6	ns	After Clock Edges(1)
t_8	PCHK# Valid Delay	3	35	12.4	ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	35	12.5	ns	
t_9	BLAST#, PLOCK# Float Delay		42	12.6	ns	After Clock Edges(1)
t_{10}	D0–D31, DP0–DP3 Write Data Valid Delay	3	30	12.5	ns	
t_{11}	D0–D31, DP0–DP3 Write Data Float Delay		42	12.6	ns	After Clock Edges(1)
t_{12}	EADS# Setup Time	12		12.2	ns	
t_{13}	EADS# Hold Time	3		12.2	ns	
t_{14}	KEN#, BS16#, BS8# Setup Time	12		12.2	ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		12.2	ns	
t_{16}	RDY#, BRDY# Setup Time	12		12.2	ns	

PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

16 MHz 3.3V Intel486™ SX Microprocessor A.C. Characteristics

V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified. (Continued)

Symbol	Parameter	Min	Max	Figure	Unit	Notes
t_{17}	RDY#, BRDY# Hold Time	3		12.2	ns	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	12		12.2	ns	
t_{19}	HOLD, AHOLD, BOFF# Setup Time	3		12.2	ns	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR Setup Time	14		12.2	ns	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR Hold Time *	3		12.2	ns	
t_{22}	D0–D31 , DP0–DP3, A4–A31 Read Setup Time	10		12.2	ns	
t_{23}	D0–D31 , DP0–DP3, A4–A31 Read Hold Time	3		12.2	ns	
t_{24}	TCK Frequency		16	12.7	MHz	1X Clock
t_{25}	TCK Period	62.5		12.7	ns	(Note 3)
t_{26}	TCK High Time	10		12.7	ns	@2.0V
t_{27}	TCK Low Time	10		12.7	ns	@0.8V
t_{28}	TCK Rise Time		4	12.7	ns	(Note 2)
t_{29}	TCK Fall Time		4	12.7	ns	(Note 2)
t_{30}	TDI, TMS Setup Time	8		12.7	ns	(Note 4)
t_{31}	TDI, TMS Hold Time	10		12.7	ns	(Note 4)
t_{32}	TDO Valid Delay	3	25	12.7	ns	(Note 4)
t_{33}	TDO Float Delay		TBD	12.7	ns	(Note 4)
t_{34}	All Outputs (Non-Test Valid Delay)	3	25	12.7	ns	(Note 4)
t_{35}	All Outputs (Non-Test Float Delay)		36	12.7	ns	(Note 4)
t_{36}	All Inputs (Non-Test) Setup Time	8		12.7	ns	(Note 4)
t_{37}	All Inputs (Non-Test) Hold Time	10		12.7	ns	(Note 4)

* PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. Rise/Fall times are measured between 0.8V and 2.0V. Rise/Fall times can be relaxed by 1 ns per 10 ns increase in TCK period.
3. TCK period ≥ CLK period.
4. Parameter measured from TCK. Boundary scan tested at V_{CC} greater than or equal to 3.3V.

20 MHz 3.3V Intel486™ SX Microprocessor A.C. Characteristics

V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Figure	Unit	Notes
	Frequency	8	20		MHz	1X Clock
t_1	CLK Period	50	125	12.1	ns	
t_{1a}	CLK Period Stability		0.1%	12.1	ns	Adjacent Clocks
t_2	CLK High Time	16		12.1	ns	at 2V
t_3	CLK Low Time	16		12.1	ns	at 0.8V
t_4	CLK Fall Time		6	12.1	ns	2V to 0.8V
t_5	CLK Rise Time		6	12.1	ns	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Valid Delay	3	23	12.5	ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		37	12.6	ns	After Clock Edges(1)
t_8	PCHK# Valid Delay	3	28	12.4	ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	28	12.5	ns	
t_9	BLAST#, PLOCK# Float Delay		37	12.6	ns	After Clock Edges(1)
t_{10}	D0–D31 , DP0–DP3 Write Data Valid Delay	3	26	12.5	ns	
t_{11}	D0–D31 , DP0–DP3 Write Data Float Delay		37	12.6	ns	After Clock Edges(1)
t_{12}	EADS# Setup Time	10		12.2	ns	
t_{13}	EADS# Hold Time	3		12.2	ns	
t_{14}	KEN#, BS16#, BS8# Setup Time	10		12.2	ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		12.2	ns	
t_{16}	RDY#, BRDY# Setup Time	10		12.2	ns	
t_{17}	RDY#, BRDY# Hold Time	3		12.2	ns	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	12		12.2	ns	
t_{19}	HOLD, AHOLD, BOFF# Setup Time	3		12.2	ns	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR Setup Time	12		12.2	ns	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR Hold Time	3		12.2	ns	
t_{22}	D0–D31 , DP0–DP3, A4–A31 Read Setup Time	8		12.2	ns	
t_{23}	D0–D31 , DP0–DP3, A4–A31 Read Hold Time	3		12.2	ns	
t_{24}	TCK Frequency		20	12.7	MHz	1X Clock
t_{25}	TCK Period	50		12.7	ns	(Note 3)
t_{26}	TCK High Time	10		12.7	ns	@ 2.0V
t_{27}	TCK Low Time	10		12.7	ns	@0.8V
t_{28}	TCK Rise Time		4	12.7	ns	(Note 2)
t_{29}	TCK Fall Time		4	12.7	ns	(Note 2)

PRELIMINARY
SEE INTEL FOR DESIGN-IN
INFORMATION

20 MHz 3.3V Intel486™ SX Microprocessor A.C. Characteristics

V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified. (Continued)

Symbol	Parameter	Min	Max	Figure	Unit	Notes
t_{30}	TDI, TMS Setup Time	8		12.7	ns	(Note 4)
t_{31}	TDI, TMS Hold Time	10		12.7	ns	(Note 4)
t_{32}	TDO Valid Delay	3	25	12.7	ns	(Note 4)
t_{33}	TDO Float Delay		TBD	12.7	ns	(Note 4)
t_{34}	All Outputs (Non-Test Valid Delay)	3	25	12.7	ns	(Note 4)
t_{35}	All Outputs (Non-Test Float Delay)		36	12.7	ns	(Note 4)
t_{36}	All Inputs (Non-Test) Setup Time	8		12.7	ns	(Note 4)
t_{37}	All Inputs (Non-Test) Hold Time	10		12.7	ns	(Note 4)

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. Rise/Fall times are measured between 0.8V and 2.0V. Rise/Fall times can be relaxed by 1 ns per 10 ns increase in TCK period.
3. TCK period ≥ CLK period.
4. Parameter measured from TCK. Boundary scan tested at V_{CC} greater than or equal to 3.3V.

PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

25 MHz 3.3V Intel486™ SX Microprocessor A.C. Characteristics

V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Figure	Unit	Notes
	Frequency	8	25		MHz	1X Clock
t_1	CLK Period	40	125	12.1	ns	
t_{1a}	CLK Period Stability		0.1%	12.1	ns	Adjacent Clocks
t_2	CLK High Time	14		12.1	ns	at 2V
t_3	CLK Low Time	14		12.1	ns	at 0.8V
t_4	CLK Fall Time		4	12.1	ns	2V to 0.8V
t_5	CLK Rise Time		4	12.1	ns	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Valid Delay	3	20	12.5	ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		28	12.6	ns	After Clock Edges(1)
t_8	PCHK# Valid Delay	3	24	12.4	ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	24	12.5	ns	
t_9	BLAST#, PLOCK# Float Delay		28	12.6	ns	After Clock Edges(1)
t_{10}	D0–D31 , DP0–DP3 Write Data Valid Delay	3	20	12.5	ns	
t_{11}	D0–D31, DP0–DP3 Write Data Float Delay		28	12.6	ns	After Clock Edges(1)
t_{12}	EADS# Setup Time	8		12.2	ns	
t_{13}	EADS# Hold Time	3		12.2	ns	

PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

25 MHz 3.3V Intel486™ SX Microprocessor A.C. Characteristics

V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified. (Continued)

Symbol	Parameter	Min	Max	Figure	Unit	Notes
t_{14}	KEN#, BS16#, BS8# Setup Time	8		12.2	ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		12.2	ns	
t_{16}	RDY#, BRDY# Setup Time	8		12.2	ns	
t_{17}	RDY#, BRDY# Hold Time	3		12.2	ns	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	10		12.2	ns	
t_{19}	HOLD, AHOLD, BOFF# Setup Time	3		12.2	ns	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR Setup Time	10		12.2	ns	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR Hold Time	3		12.2	ns	
t_{22}	D0–D31, DP0–DP3, A4–A31 Read Setup Time	8		12.2	ns	
t_{23}	D0–D31, DP0–DP3, A4–A31 Read Hold Time	3		12.2	ns	
t_{24}	TCK Frequency		25	12.7	MHz	1X Clock
t_{25}	TCK Period	40		12.7	ns	(Note 3)
t_{26}	TCK High Time	10		12.7	ns	@2.0V
t_{27}	TCK Low Time	10		12.7	ns	@0.8V
t_{28}	TCK Rise Time		4	12.7	ns	(Note 2)
t_{29}	TCK Fall Time		4	12.7	ns	(Note 2)
t_{30}	TDI, TMS Setup Time	8		12.7	ns	(Note 4)
t_{31}	TDI, TMS Hold Time	10		12.7	ns	(Note 4)
t_{32}	TDO Valid Delay	3	25	12.7	ns	(Note 4)
t_{33}	TDO Float Delay		TBD	12.7	ns	(Note 4)
t_{34}	All Outputs (Non-Test Valid Delay)	3	25	12.7	ns	(Note 4)
t_{35}	All Outputs (Non-Test Float Delay)		36	12.7	ns	(Note 4)
t_{36}	All Inputs (Non-Test) Setup Time	8		12.7	ns	(Note 4)
t_{37}	All Inputs (Non-Test) Hold Time	10		12.7	ns	(Note 4)

PRELIMINARY
FOR DESIGN-IN
INFORMATION
SEE INTEL

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. Rise/Fall times are measured between 0.8V and 2.0V. Rise/Fall times can be relaxed by 1 ns per 10 ns increase in TCK period.
3. TCK period ≥ CLK period.
4. Parameter measured from TCK. Boundary scan tested at V_{CC} greater than or equal to 3.3V.

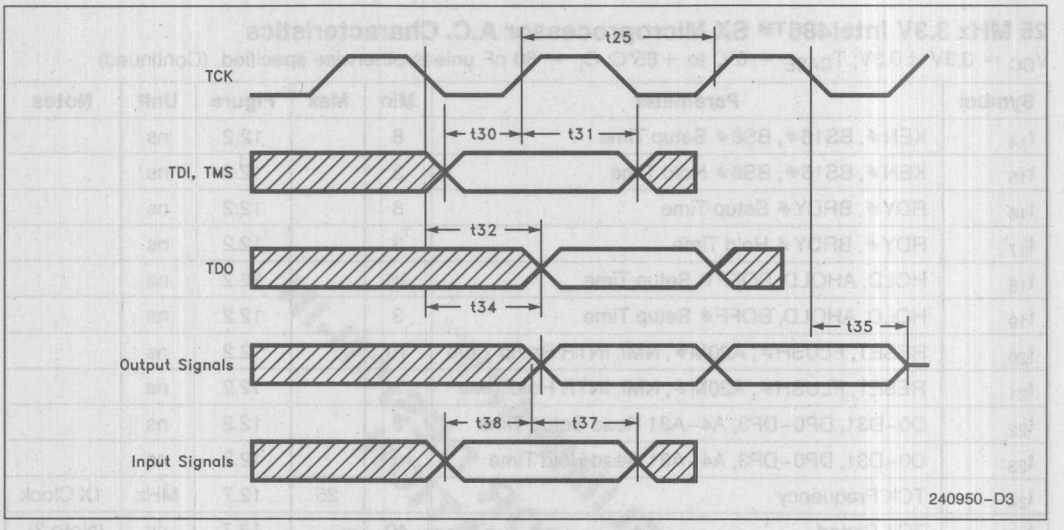


Figure 1. Test Signal Timing Diagram

Intel486™ Microprocessor Family Data Sheet Addendum: SL Enhanced Intel486™ Microprocessor Family

November 1993

Order Number: 241696-002

SL Enhanced Intel486 Microprocessor Family

CONTENTS

PAGE

1.0. INTRODUCTION 2-719

1.1. SL Enhanced Intel486 Microprocessor Features 2-719

1.2. The SL Enhanced Intel486 CPU Product Family 2-721

2.0. PIN DESCRIPTION 2-721

2.1. Pin Assignments 2-721

2.2. Quick Pin Reference 2-731

3.0. CLOCK CONTROL 2-733

3.1. Clock Generation 2-733

3.2. Stop Clock 2-733

3.3. Stop Grant Bus Cycle 2-734

3.4. Pin State during Stop Grant 2-734

3.5. Clock Control State Diagram 2-735

3.5.1. Normal State 2-735

3.5.2. Stop Grant State 2-735

3.5.3. Stop Clock State 2-736

3.5.4. Auto Halt Powerdown State 2-737

3.5.5. Stop Clock Snoop State (Cache Invalidations) 2-737

3.5.6. Auto Idle Powerdown State 2-737

3.6. Supply Current Model for Stop Clock Modes and Transitions 2-737

4.0. INTEL'S SYSTEM MANAGEMENT MODE ARCHITECTURE 2-738

4.1. SMM Overview 2-738

4.2.Terminology 2-738

4.3. System Management Interrupt Processing 2-739

4.3.1. System Management Interrupt (SMI#) 2-739

4.3.2. SMI Active (SMIACT#) 2-741

4.3.3. SMRAM 2-742

4.3.3.1. SMRAM State Save Map 2-742

4.3.4. Exit from SMM 2-744

CONTENTS

PAGE

4.4. System Management Mode Programming Model 2-744

4.4.1. Entering System Management Mode 2-744

4.4.2. Processor Environment 2-745

4.4.3. Executing System Management Mode Handler 2-746

4.4.3.1. Exceptions and Interrupts within System Management Mode 2-746

4.5. SMM Features 2-747

4.5.1. SMM Revision Identifier 2-747

4.5.2. Halt Auto Restart 2-748

4.5.3. I/O Instruction Restart 2-748

4.5.4. SMM Base Relocation 2-748

4.6. SMM-System Design Considerations 2-749

4.6.1. SMRAM Interface 2-749

4.6.2. Cache Flushes 2-750

4.6.3. A20M# Pin 2-751

4.6.4. CPU Reset during SMM 2-751

4.6.5. SMM and Second Level Write Buffers 2-752

4.6.6. Nested SMI#s and I/O Restart 2-752

4.7. SMM-Software Considerations 2-752

4.7.1. SMM Code Considerations 2-752

4.7.2. Exception Handling 2-752

4.7.3. Halt during SMM 2-753

4.7.4. Relocating SMRAM to an Address above One Megabyte 2-753

5.0. RESET AND INITIALIZATION 2-753

5.1. RESET 2-753

5.2. SRESET 2-753

CONTENTS

PAGE

5.3. Pin State During Reset 2-753
5.4. CPU Identification Codes 2-755
5.4.1 CPU ID Instruction 2-755

6.0. ELECTRICAL AND MECHANICAL SPECIFICATIONS 2-757
6.1. D.C. Specifications 2-757
6.1.1. 3.3V D.C. Characteristics ... 2-757
6.1.2. 5V D.C. Characteristics 2-759
6.2. A.C. Specifications for 1X CLK Option 2-761
6.2.1. 5V A.C. Characteristics 2-761
6.2.2. 3.3V A.C. Characteristics ... 2-765
6.3. Mechanical Data 2-767
6.3.1. Package Mechanical Specifications for the 208-Lead SQFP Package 2-767
6.3.2. Package Thermal Specifications 2-768
6.4. Capacitive Derating Information 2-771

7.0. 2X CLOCK MODE 2-774
7.1. Pin Assignments 2-774
7.2. Quick Pin Reference 2-775
7.3. Clock Control 2-776
7.3.1. Clock Generation 2-776
7.3.2. Stop Clock 2-776
7.3.3. Clock Control State Diagram 2-777
7.3.3.1. Normal State 2-777
7.3.3.2. Stop Grant State 2-777
7.3.3.3. Stop Clock State 2-778
7.3.3.4. HALT State 2-778
7.3.4. Supply Current Model for Stop Clock Modes and Transitions 2-779
7.4. D.C. Specifications for 2X Clock Option 2-780
7.5. A.C. Specifications for 2X Clock Option 2-780
7.5.1. 5V A.C. Characteristics 2-781
7.5.2. 3.3V A.C. Characteristics ... 2-782

CONTENTS

PAGE

8.0. TESTABILITY 2-784
8.1. Test Access Port 2-784
8.2. Boundary Scan Component Identification Support 2-785

9.0. OverDrive™ PROCESSOR SOCKET FOR SL ENHANCED Intel486 CPU-BASED SYSTEMS 2-786
9.1. OverDrive Processor Socket for 5V SL Enhanced Intel486 SX and DX CPU-Based Systems 2-786
9.1.1. Overdrive Processor Socket Circuit Design 2-786
9.1.1.1 OverDrive Processor Socket Circuit for SL Enhanced Intel486 DX CPU-Based Systems 2-787
9.1.1.2 OverDrive Processor Socket Circuit for SL Enhanced Intel486 SX CPU-Based Systems 2-787
9.1.2. Socket Layout 2-788
9.1.3. Thermal Management 2-789
9.1.4. Design Considerations 2-789
9.1.5. Testability 2-790
9.1.6. Overdrive Processor Socket Pinout for 5V Intel486 SX and DX CPU-Based Systems 2-790
9.1.7. D.C./A.C. Characteristics ... 2-792
9.2. Pentium™ OverDrive Processor Socket for 5V SL Enhanced Intel486 DX2 CPU-Based Systems 2-792
9.2.1. Pentium™ OverDrive Processor Socket Overview for 5V SL Enhanced Intel486 DX2 CPU-Based Systems 2-793
9.2.2. Mechanical Design Considerations 2-793
9.2.3. Thermal Design Considerations 2-794
9.2.4. Pentium™ OverDrive Processor Socket Pinout for 5V SL Enhanced Intel486 DX2 CPU-Based Systems 2-795
9.2.5. D.C./A.C. Characteristics ... 2-797

10.0 REVISION HISTORY 2-798

CONTENTS

PAGE

APPENDIX A—SYSTEM DESIGN NOTES 2-799

A.1. SMM Environment Initialization .. 2-799

A.2. Accessing SMRAM 2-801

A.2.1. Loading SMRAM with an Initial SMI Handler 2-801

A.2.2. SMRAM Hidden from DMA and BUS Masters 2-801

A.2.3. Accessing System Memory from within SMM 2-802

A.3. Interrupts and Exceptions during SMM Handler Routines 2-803

A.3.1. SMM Compliant Vector Tables 2-803

CONTENTS

PAGE

A.3.2. Interrupts and Subroutines with SMRAM Relocation 2-804

A.4. Floating Point Operation and SMM 2-804

A.4.1. The Need to Save the FPU Environment 2-804

A.4.2. Saving the State of the Floating Point Unit 2-804

A.5. Support for Power Managed Peripherals 2-806

A.5.1. Shadow Registers 2-806

A.5.2. Handling Interrupted I/O Write Sequences 2-807

1.0 INTRODUCTION

Intel is enhancing its entire Intel486™ microprocessor family with the energy-efficient technology driving today's highly integrated Intel486 SL CPUs. The SL Enhanced Intel486 microprocessor family enables built-in power management while maintaining full compatibility with existing Intel architecture processors. The SL Enhanced Intel486 CPU family allows system designers to design desktop systems that exceed the Environmental Protection Agency's (EPA) Energy Star guidelines, without sacrificing performance. Bringing SL Technology to the entire Intel486 CPU line increases notebook system design flexibility and increases the battery life of all high-performance Intel486 CPU-based notebooks. SL technology allows system designers to differentiate their power management schemes with a variety of energy-efficient or battery life-preserving features.

SL Enhanced Intel486 CPU-based systems can maximize both energy efficiency and performance. SL Technology features like **Stop Clock** and **Auto HALT** power down allow CPU standby modes, saving systems up to an additional 4W over previous Intel486 CPUs while providing virtually instantaneous recovery to a full-on state. **Stop Clock** and **Auto Idle** power down also allow active power management of the CPU so systems can save energy even when in use. SL Enhanced OverDrive™ processors provide end-user performance upgradability to SL Enhanced Intel486 CPUs while maintaining their energy efficiency.

SL Enhanced Intel486 CPUs enable power management features to be built into hardware, allowing energy efficiency that is transparent to application and O/S software. **Stop Clock**, **Auto HALT** power down, and **Auto Idle** power down allow software transparent control over CPU power management. Equally important is the capability of the processor to manage system power consumption. The SL Enhanced Intel486 CPU incorporates the same **System Management Mode (SMM)** found in the SL CPU family. A non-maskable **System Management Interrupt (SMI)**, a corresponding **Resume (RSM)** instruction and a new memory space for system management code are the basis of Intel's System Management Mode. Intel's SMM ensures seamless power control of the processor core, system logic, main memory and one or more peripheral devices transparent to any application or operating system.

SL Enhanced Intel486 CPUs are available in a full range of speeds (25 MHz to 66 MHz), packages (PGA, SQFP, PQFP), and voltages (5V, 3.3V) to meet any system design requirement.

This Data Book Addendum highlights the features of Intel's SL Enhanced Intel486 microprocessors and should be used in conjunction with the latest version of the existing Intel486 microprocessor documents, including the following:

- *Intel486™ DX Microprocessor Data Book*, Order No. 240440
- *Intel486™ SX Microprocessor-Intel487™ SX Co-Processor Data Sheet*, Order No. 240950
- *Intel486™ Family of Microprocessors Low Power Version* Data Sheet, Order No. 241199
- *Intel486™ Microprocessor Family Programmer's Reference Manual*, Order No. 240486
- *Intel486™ Microprocessor Hardware Reference Manual*, Order No. 240552
- *Intel486™ DX2 Microprocessor Data Book*, Order No. 241245
- *Pentium™ OverDrive™ Processor Data Sheet*, Order No. 290436
- 1993 Packaging Handbook, Order No. 240800

End-user upgradability for the SL Enhanced Intel486 CPU product family is specified in the OverDrive Processor socket section (Section 9) of this document.

1.1 SL Enhanced Intel486 Microprocessor Features

- *Intel's System Management Mode* — A unique Intel architecture operating mode with its dedicated special purpose interrupt and address space which can be used to implement intelligent power management and other enhanced functions in a manner which is completely transparent to the operating system and applications software.
- *I/O Restart* — An I/O instruction interrupted by a System Management Interrupt (SMI#) can automatically be re-started during the execution of an RSM instruction.
- *Stop Clock* — Provides a clock control mechanism for the SL Enhanced Intel486 CPUs. This mechanism provides two low-power states: a fast wake-up Stop Grant State (~20–55 mA) and a Stop Clock state with **CLK frequency at 0-MHz (~100 μA–200 μA)**.
- *Auto HALT Power Down* — After the execution of a HALT instruction, the CPU issues a normal HALT bus cycle and the clock input to the SL Enhanced Intel486 CPU core is automatically stopped, causing the CPU to enter the Auto HALT Power Down state (~20 mA–55 mA).
- *Auto Idle Power Down* — This function which only applies to Intel486 DX2 CPUs, allows the CPU to reduce its core clock rate to half of its original frequency, without affecting performance. Auto Idle Power Down provides an average power savings of 10%.

- *Upgrade Power Down Mode* — When an OverDrive processor upgrade is installed, this feature detects the presence of the upgrade, then powers down the core, and three-states all outputs of the original CPU, so the original CPU consumes very low current.
- *Package Options* — The SL Enhanced Intel486 CPUs are available in 168 lead PGA, 196 lead PQFP, and 208 lead SQFP packages. The SQFP package provides a 40% volume savings over PQFP, making it an ideal solution for mobile systems.
- *3.3 Volt Operation* — The SL Enhanced Intel486 CPUs are available with either a 3.3V supply voltage or a 5V supply voltage. The 3.3V supply voltage provides a 50% power saving over a 5V supply voltage.
- *Clocking Options* — The SL Enhanced Intel486 CPUs are available with either a 1X clock input or a 2X clock input. The 1X clock option, in which the phases of the core clock are provided by an internal Phase Lock Loop circuit, provides a simpler system design with better I/O-timing performance. The 1X clock option, is required for DX2 clock-doubled systems and OverDrive Processor upgradability. The 2X clock option, in which both phases of the core clock are provided by the system, allows portable designs to use existing Intel486 "LP" CPU clock control mechanisms. The 2X clock option is intended as a short-term solution for portable applications.

All of the SL Enhanced Intel486 CPUs are based on an **Intel486 CPU** core which consists of a 32-bit integer processing unit, an 8 Kbyte cache, and a memory management unit. This ensures full binary compatibility with the 8086, 8088, 80186, 80286, Intel386™ SX, Intel386 DX, and all versions of Intel486 microprocessors. All of the Intel486 microprocessors described in this document offer the following features:

- *Full 32-bit RISC integer processor* — The Intel486 performs a complete set of arithmetic and logical operations on 8-, 16-, and 32-bit data types using a full-width ALU and eight general purpose registers.
- *Single Cycle Execution* — Many instructions execute in a single clock cycle.
- *Instruction Pipelining* — The fetching, decoding, execution, and address translation of instructions is overlapped within the Intel486 microprocessor.
- *On-Chip Floating Point Unit* — (All except Intel486 SX CPU) Supports 32-, 64-, and 80-bit formats specified in IEEE standard 754 are supported. The unit is binary compatible with the 8087, 80287, Intel387™SX and DX, and Intel487™ SX math coprocessors.
- *On-Chip Cache with Cache Consistency Support* — An 8 Kbyte internal write-through cache is used for both data and instructions. Cache hits provide zero wait-state access times for data within the cache. Bus activity is tracked to detect alterations in the memory which the internal cache represents. The internal cache can be invalidated or flushed so that an external cache controller can maintain cache consistency in multi-processor environments.
- *External Cache Control* — Write-back and flush controls for an external L2 cache are provided so the processor can maintain cache consistency in multiprocessor environments.
- *On-Chip Memory Management Unit* — Address management and memory space protection mechanisms maintain the integrity of memory in a multitasking and virtual memory environment. Both segmentation and paging are supported.
- *Separate 32-bit Address and Data Paths* — Up to four gigabytes of physical memory can be addressed directly.
- *Burst Cycles* — Burst transfers allow a new double word to be read from memory on each clock cycle. This capability is especially useful for instruction prefetch and for filling the internal cache.
- *Write Buffers* — The processor contains four write buffers to enhance the performance of consecutive writes to memory. The CPU can continue internal operations after a write, without waiting for the write to be executed on the external bus.
- *Bus Backoff* — If another bus master needs control of the bus during a CPU initiated bus cycle, the Intel486 CPU will float its bus signals, then restart the cycle when the bus becomes available again.
- *Instruction Restart* — Programs can continue execution following an exception generated by an unsuccessful attempt to access memory. This feature is important for supporting demand-paged virtual memory applications.
- *Dynamic Bus Sizing* — External controllers can dynamically alter the effective width of the data bus. Bus widths of 8, 16, or 32 bits can be used.
- *Boundary Scan (JTAG)* — Boundary Scan provides for in-circuit testing of components on printed circuit boards. The Intel Boundary Scan implementation conforms with the IEEE Standard Test Access Port and Boundary Scan Architecture. Not all products are offered with the boundary scan feature.

1.2 The SL Enhanced Intel486 CPU Product Family

Table 1-1 shows the SL Enhanced Intel486 CPUs and existing Intel486 microprocessors available by Clock Mode, Supply Voltage, Maximum Frequency, and Package. An individual product will have either a 1X clock or a 2X clock, but not both. Likewise, an individual product will have either a 5V supply voltage or a 3.3V supply voltage, but not both. Please contact Intel for the latest product availability and specifications.

Table 1-1. Product Options

Existing Intel486 CPUs	SL Enhanced Intel486 CPUs	V_{CC}	CPU FREQUENCY						168 PGA	196 PQFP	208 SQFP
1X CLOCK											
Intel486 SX CPU	SL Enhanced Intel486 SX CPU	3.3V		25	33						X
		5V		25	33				X	X	
Intel486 DX CPU	SL Enhanced Intel486 DX CPU	3.3V			33						X
		5V			33				X	X	
							50		X		
Intel486 DX2 CPU	SL Enhanced Intel486 DX2 CPU	3.3V				40	50				X
		5V					50	66	X		
2X CLOCK											
Low Power Intel486 SX CPU	SL Enhanced Intel486 SX CPU	3.3V		25	33						X
		5V		25	33					X	
Low Power Intel486 DX CPU	SL Enhanced Intel486 DX CPU	3.3V			33						X
		5V			33					X	

2.0 PIN DESCRIPTION

2.1 Pin Assignments

The following figures show the pin assignments of each package type for the SL Enhanced Intel486 CPU product family. Tables are provided showing the pin differences between the existing Intel486 CPU products and the SL Enhanced Intel486 CPU products.

NOTE:
NC pins should ALWAYS remain unconnected.

196 Lead PQFP — Plastic Quad Flat Pack
— Package Diagram
— Pin Assignment Difference Table
— Pin Assignment Table in numerical order

168 Lead PGA — Pin Grid Array
— Package Diagram
— Pin Assignment Difference Table
— Pin Cross Reference by Pin Name

208 Lead SQFP — Quad Flat Pack
— Package Diagram
— Pin Assignment Difference Table
— Pin Assignment Table in numerical order

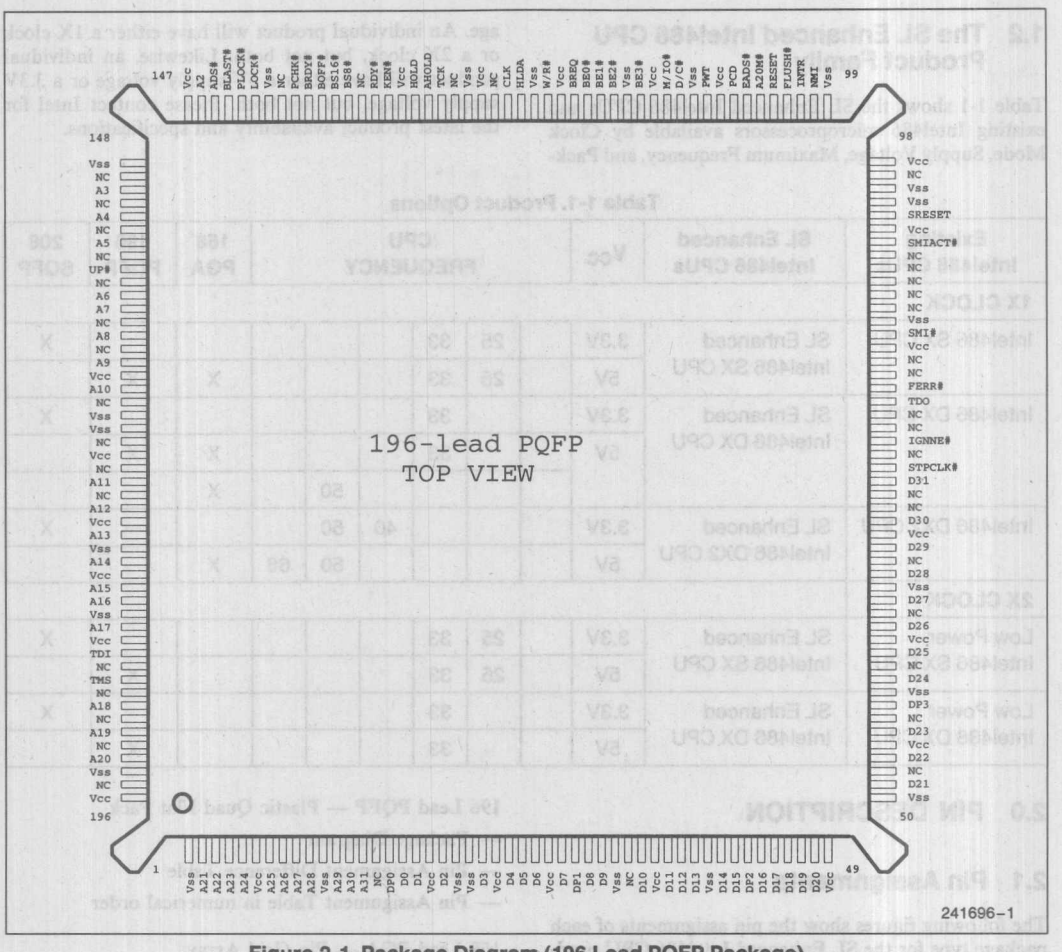


Figure 2-1. Package Diagram (196 Lead PQFP Package)

Table 2-1. Pinout Differences for 196 Lead PQFP Package

Pin #	Intel486 SX CPU	Low Power Intel486 SX CPU	SL Enhanced Intel486 SX CPU	Intel486 DX CPU	Low Power Intel486 DX CPU	SL Enhanced Intel486 DX CPU
75	NC	NC	STPCLK#	NC	NC	STPCLK#
77	NC	NC	NC	IGNNE#	IGNNE#	IGNNE#
81	NC	NC	NC	FERR#	FERR#	FERR#
85	NC	NC	SMI#	NC	NC	SMI#
92	NC	NC	SMIACT#	NC	NC	SMIACT#
94	NC	NC	SRESET	NC	NC	SRESET
127	NC	CLKSEL	NC	NC	CLKSEL	NC

Table 2-2. Pin Assignments for 196 Lead PQFP Package

Pin#	Description	Pin#	Description	Pin#	Description	Pin#	Description
1	V_{SS}	38	D12	75	STPCLK#	112	V_{CC}
2	A21	39	D13	76	NC	113	BE3#
3	A22	40	V_{SS}	77	IGNNE#	114	V_{SS}
4	A23	41	D14	78	NC	115	BE2#
5	A24	42	D15	79	NC	116	BE1#
6	V_{CC}	43	DP2	80	TDO	117	BE0#
7	A25	44	D16	81	FERR#	118	BREQ
8	A26	45	D17	82	NC	119	V_{CC}
9	A27	46	D18	83	NC	120	W/R#
10	A28	47	D19	84	V_{CC}	121	V_{SS}
11	V_{SS}	48	D20	85	SMI#	122	HLDA
12	A29	49	V_{CC}	86	V_{SS}	123	CLK
13	A30	50	V_{SS}	87	NC	124	NC
14	A31	51	D21	88	NC	125	V_{CC}
15	NC	52	NC	89	NC	126	V_{SS}
16	DP0	53	D22	90	NC	127	NC
17	D0	54	V_{CC}	91	NC	128	TCK
18	D1	55	D23	92	SMIACT#	129	AHOLD
19	V_{CC}	56	NC	93	V_{CC}	130	HOLD
20	D2	57	DP3	94	SRESET	131	V_{CC}
21	V_{SS}	58	V_{SS}	95	V_{SS}	132	KEN#
22	V_{SS}	59	D24	96	V_{SS}	133	RDY#
23	D3	60	NC	97	NC	134	NC
24	V_{CC}	61	D25	98	V_{CC}	135	BS8#
25	D4	62	V_{CC}	99	V_{SS}	136	BS16#
26	D5	63	D26	100	NMI	137	BOFF#
27	D6	64	NC	101	INTR	138	BRDY#
28	V_{CC}	65	D27	102	FLUSH#	139	PCHK#
29	D7	66	V_{SS}	103	RESET	140	NC
30	DP1	67	D28	104	A20M#	141	V_{SS}
31	D8	68	NC	105	EADS#	142	LOCK#
32	D9	69	D29	106	PCD	143	PLOCK#
33	V_{SS}	70	V_{CC}	107	V_{CC}	144	BLAST#
34	NC	71	D30	108	PWT	145	ADS#
35	D10	72	NC	109	V_{SS}	146	A2
36	V_{CC}	73	NC	110	D/C#	147	V_{CC}
37	D11	74	D31	111	M/IO#	148	V_{SS}

Table 2-2. Pin Assignments for 196 Lead PQFP Package (Continued)

Pin #	Description	Pin #	Description	Pin #	Description	Pin #	Description
149	NC	161	A8	173	NC	185	TDI
150	A3	162	NC	174	A12	186	NC
151	NC	163	A9	175	V_{CC}	187	TMS
152	A4	164	V_{CC}	176	A13	188	NC
153	NC	165	A10	177	V_{SS}	189	A18
154	A5	166	NC	178	A14	190	NC
155	NC	167	V_{SS}	179	V_{CC}	191	A19
156	UP#	168	V_{SS}	180	A15	192	NC
157	NC	169	NC	181	A16	193	A20
158	A6	170	V_{CC}	182	V_{SS}	194	V_{SS}
159	A7	171	NC	183	A17	195	NC
160	NC	172	A11	184	V_{CC}	196	V_{CC}

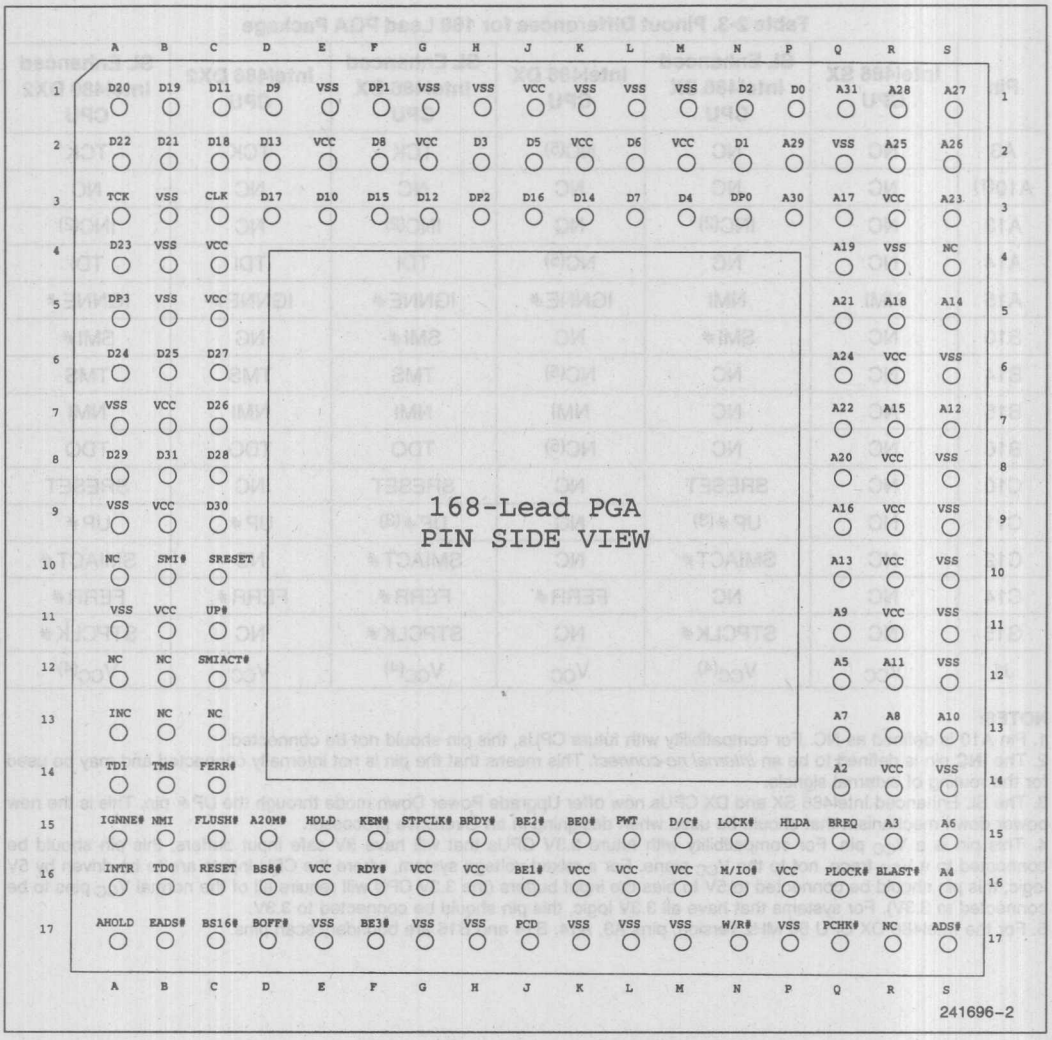


Figure 2-2. Package Diagram for 168 Lead PGA Package

Table 2-3. Pinout Differences for 168 Lead PGA Package

Pin	Intel486 SX CPU	SL Enhanced Intel486 SX CPU	Intel486 DX CPU	SL Enhanced Intel486 DX CPU	Intel486 DX2 CPU	SL Enhanced Intel486 DX2 CPU
A3	NC	NC	NC(5)	TCK	TCK	TCK
A10(1)	NC	NC	NC	NC	NC	NC
A13	NC	INC(2)	NC	INC(2)	NC	INC(2)
A14	NC	NC	NC(5)	TDI	TDI	TDI
A15	NMI	NMI	IGNNE#	IGNNE#	IGNNE#	IGNNE#
B10	NC	SMI#	NC	SMI#	NC	SMI#
B14	NC	NC	NC(5)	TMS	TMS	TMS
B15	NC	NC	NMI	NMI	NMI	NMI
B16	NC	NC	NC(5)	TDO	TDO	TDO
C10	NC	SRESET	NC	SRESET	NC	SRESET
C11	NC	UP#(3)	NC	UP#(3)	UP#	UP#
C12	NC	SMIACT#	NC	SMIACT#	NC	SMIACT#
C14	NC	NC	FERR#	FERR#	FERR#	FERR#
G15	NC	STPCLK#	NC	STPCLK#	NC	STPCLK#
J1	V_{CC}	V_{CC}(4)	V_{CC}	V_{CC}(4)	V_{CC}	V_{CC}(4)

NOTES:
1. Pin A10 is defined as NC. For compatibility with future CPUs, this pin should not be connected.
2. The **INC** pin is defined to be an *internal no-connect*. This means that the pin is not internally connected and may be used for the routing of external signals.
3. The SL Enhanced Intel486 SX and DX CPUs now offer Upgrade Power Down mode through the UP# pin. This is the new power down mechanism that should be used when designing in an OverDrive processor.
4. This pin is a V_{CC} pin. For compatibility with future 3.3V CPUs that will have 5V safe input buffers, this pin should be connected to a V_{CC} trace, not to the V_{CC} plane. For a mixed voltage system, where the CPU inputs are to be driven by 5V logic, this pin should be connected to 5V to bias the input buffers (the 3.3V CPU will require all of the normal V_{CC} pins to be connected to 3.3V). For systems that have all 3.3V logic, this pin should be connected to 3.3V.
5. For the Intel486 DX CPU 50 MHz version, pins A3, A14, B14 and B16 are boundary scan pins.

Table 2-4. Pin Cross Reference for 168 Lead PGA Package

Address		Data		Control		NC	V_{CC}	V_{SS}
A2	Q14	D0	P1	A20M#	D15	A10(1)	B7	A7
A3	R15	D1	N2	ADS#	S17	A13(2)	B9	A9
A4	S16	D2	N1	AHOLD	A17	B12	B11	A11
A5	Q12	D3	H2	BE0#	K15	B13	C4	B3
A6	S15	D4	M3	BE1#	J16	C13	C5	B4
A7	Q13	D5	J2	BE2#	J15	R17	E2	B5
A8	R13	D6	L2	BE3#	F17	S4	E16	E1
A9	Q11	D7	L3	BLAST#	R16		G2	E17
A10	S13	D8	F2	BOFF#	D17		G16	G1
A11	R12	D9	D1	BRDY#	H15		H16	G17
A12	S7	D10	E3	BREQ	Q15		J1	H1
A13	Q10	D11	C1	BS8#	D16		K2	H17
A14	S5	D12	G3	BS16#	C17		K16	K1
A15	R7	D13	D2	CLK	C3		L16	K17
A16	Q9	D14	K3	D/C#	M15		M2	L1
A17	Q3	D15	F3	DP0	N3		M16	L17
A18	R5	D16	J3	DP1	F1		P16	M1
A19	Q4	D17	D3	DP2	H3		R3	M17
A20	Q8	D18	C2	DP3	A5		R6	P17
A21	Q5	D19	B1	EADS#	B17		R8	Q2
A22	Q7	D20	A1	FERR#	C14		R9	R4
A23	S3	D21	B2	FLUSH#	C15		R10	S6
A24	Q6	D22	A2	HLDA	P15			S8

Table 2-4. Pin Cross Reference for 168 Lead PGA Package (Continued)

Address		Data		Control		NC	V_{CC}	V_{SS}
A25	R2	D23	A4	HOLD	E15		R14	S9
A26	S2	D24	A6	IGNNE#	A15			S10
A27	S1	D25	B6	INTR	A16			S11
A28	R1	D26	C7	KEN#	F15			S12
A29	P2	D27	C6	LOCK#	N15			S14
A30	P3	D28	C8	M/IO#	N16			
A31	Q1	D29	A8	NMI	B15			
		D30	C9	PCD	J17			
		D31	B8	PCHK#	Q17			
				PWT	L15			
				PLOCK#	Q16			
				RDY#	F16			
				RESET	C16			
				SMI#	B10			
				SMIACT#	C12			
				UP#	C11(3)			
				W/R#	N17			
				STPCLK#	G15			
				SRESET	C10			
				TCK	A3(4)			
				TDI	A14(4)			
				TDO	B16(4)			
				TMS	B14(4)			

NOTES:
1. Pin A10. See note 1 for Table 2-3.
2. Pin A13. See note 2 for Table 2-3.
3. Pin C11. See note 3 for Table 2-3.
4. Boundary Scan pins are not included on the Intel486 SX CPU in PGA.

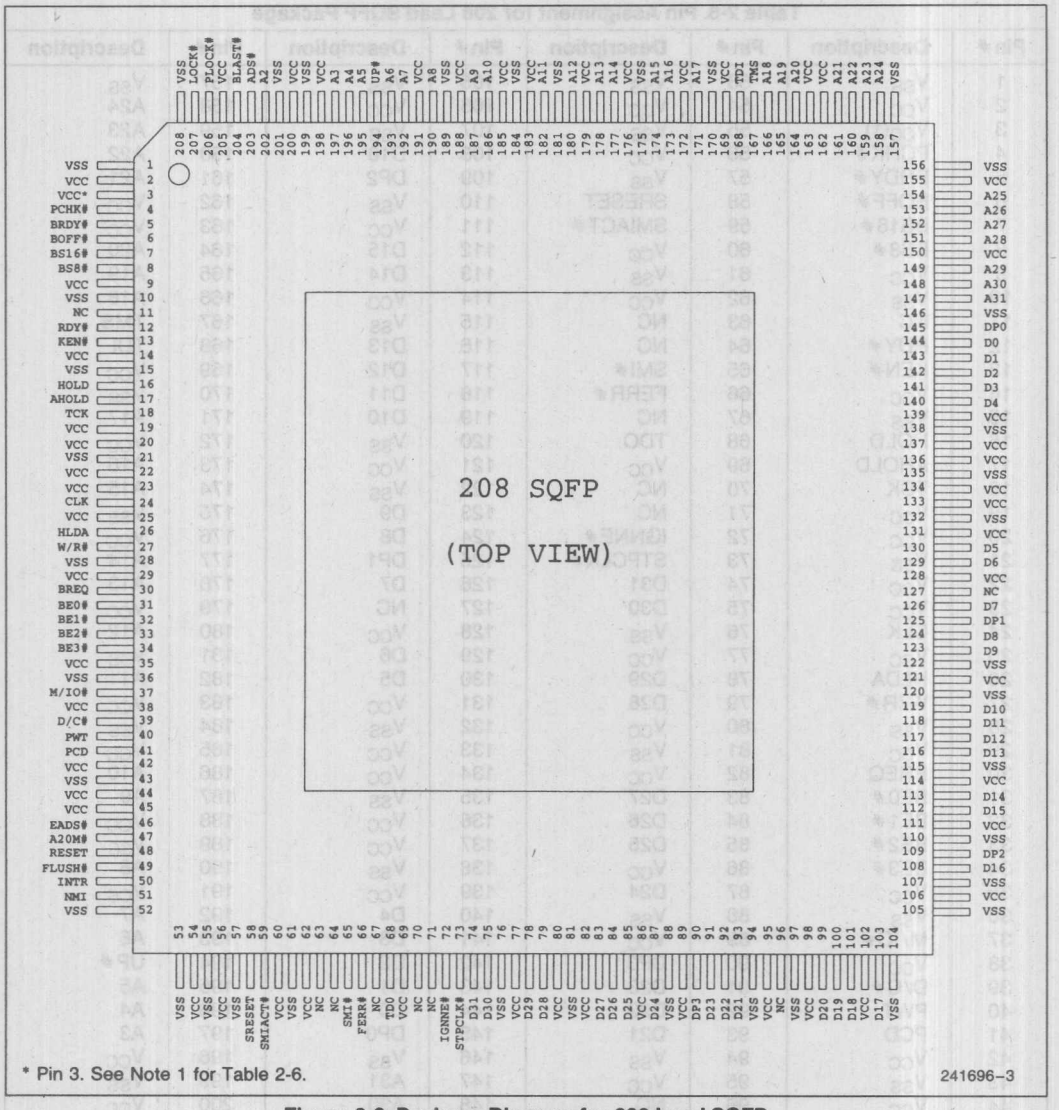


Figure 2-3. Package Diagram for 208 Lead SQFP

Table 2-5. Pinout Differences for 208 Lead SQFP Package

Pin #	SL Enhanced Intel486 SX CPU	SL Enhanced Intel486 DX CPU	SL Enhanced Intel486 DX2 CPU
66	NC	IGNNE#	IGNNE#
72	NC	FERR#	FERR#

Table 2-6. Pin Assignment for 208 Lead SQFP Package

Pin #	Description	Pin #	Description	Pin #	Description	Pin #	Description
1	V_{SS}	53	V_{SS}	105	V_{SS}	157	V_{SS}
2	V_{CC}	54	V_{CC}	106	V_{CC}	158	A24
3	V_{CC}(1)	55	V_{SS}	107	V_{SS}	159	A23
4	PCHK#	56	V_{CC}	108	D16	160	A22
5	BRDY#	57	V_{SS}	109	DP2	161	A21
6	BOFF#	58	SRESET	110	V_{SS}	162	V_{CC}
7	BS16#	59	SMIACT#	111	V_{CC}	163	V_{CC}
8	BS8#	60	V_{CC}	112	D15	164	A20
9	V_{CC}	61	V_{SS}	113	D14	165	A19
10	V_{SS}	62	V_{CC}	114	V_{CC}	166	A18
11	NC	63	NC	115	V_{SS}	167	TMS
12	RDY#	64	NC	116	D13	168	TDI
13	KEN#	65	SMI#	117	D12	169	V_{CC}
14	V_{CC}	66	FERR#	118	D11	170	V_{SS}
15	V_{SS}	67	NC	119	D10	171	A17
16	HOLD	68	TDO	120	V_{SS}	172	V_{CC}
17	AHOLD	69	V_{CC}	121	V_{CC}	173	A16
18	TCK	70	NC	122	V_{SS}	174	A15
19	V_{CC}	71	NC	123	D9	175	V_{SS}
20	V_{CC}	72	IGNNE#	124	D8	176	V_{CC}
21	V_{SS}	73	STPCLK#	125	DP1	177	A14
22	V_{CC}	74	D31	126	D7	178	A13
23	V_{CC}	75	D30	127	NC	179	V_{CC}
24	CLK	76	V_{SS}	128	V_{CC}	180	A12
25	V_{CC}	77	V_{CC}	129	D6	181	V_{SS}
26	HLDA	78	D29	130	D5	182	A11
27	W/R#	79	D28	131	V_{CC}	183	V_{CC}
28	V_{SS}	80	V_{CC}	132	V_{SS}	184	V_{SS}
29	V_{CC}	81	V_{SS}	133	V_{CC}	185	V_{CC}
30	BREQ	82	V_{CC}	134	V_{CC}	186	A10
31	BE0#	83	D27	135	V_{SS}	187	A9
32	BE1#	84	D26	136	V_{CC}	188	V_{CC}
33	BE2#	85	D25	137	V_{CC}	189	V_{SS}
34	BE3#	86	V_{CC}	138	V_{SS}	190	A8
35	V_{CC}	87	D24	139	V_{CC}	191	V_{CC}
36	V_{SS}	88	V_{SS}	140	D4	192	A7
37	M/IO#	89	V_{CC}	141	D3	193	A6
38	V_{CC}	90	DP3	142	D2	194	UP#
39	D/C#	91	D23	143	D1	195	A5
40	PWT	92	D22	144	D0	196	A4
41	PCD	93	D21	145	DP0	197	A3
42	V_{CC}	94	V_{SS}	146	V_{SS}	198	V_{CC}
43	V_{SS}	95	V_{CC}	147	A31	199	V_{SS}
44	V_{CC}	96	NC	148	A30	200	V_{CC}
45	V_{CC}	97	V_{SS}	149	A29	201	V_{SS}
46	EADS#	98	V_{CC}	150	V_{CC}	202	A2
47	A20M#	99	D20	151	A28	203	ADS#
48	RESET	100	D19	152	A27	204	BLAST#
49	FLUSH#	101	D18	153	A26	205	V_{CC}
50	INTR	102	V_{CC}	154	A25	206	PLOCK#
51	NMI	103	D17	155	V_{CC}	207	LOCK#
52	V_{SS}	104	V_{SS}	156	V_{SS}	208	V_{SS}

NOTE:
1. This pin is a V_{CC} pin. For compatibility with future 3.3V CPUs that will have 5V safe input buffers, this pin should be connected to a V_{CC} trace, not to the V_{CC} plane. For a mixed voltage system, where the CPU inputs are to be driven by 5V logic, this pin should be connected to 5V to bias the input buffers (the 3.3V CPU will require all of the normal V_{CC} pins to be connected to 3.3V). For systems that have all 3.3V logic, this pin should be connected to 3.3V.

2.2 Quick Pin Reference

Table 2-7. Pin Descriptions

Symbol	Type	Name and Function
CLK	I	**CLocK** provides the fundamental timing and the internal operating frequency for the CPU. All external timing parameters are specified with respect to the rising edge of CLK. For the 1X clock mode the CLK frequency is the same as the frequency of the CPU. For the DX2 clock doubled products the CLK frequency is half the frequency of the CPU.
RESET	I	The **RESET** input forces the CPU to begin execution at a known state. Reset is asynchronous, but must meet setup and hold times t20 and t21 for recognition in any specific clock. The CPU cnnot begin execution of instructions until at least 1 ms after V_{CC} and CLK have reached their proper AC and DC specifications. However, for soft resets, RESET should remain active for at least 15 CLK periods. The RESET pin should remain active during this time to ensure proper CPU operation. RESET is active HIGH. RESET sets the SMBASE descriptor to a default address of 30000H. If the system uses SMBASE relocation, then the SRESET pin should be used for soft resets.
SRESET	I	The SRESET pin duplicates all the functionality of the RESET pin with the following two exceptions: 1. The SMBASE register will retain its previous value. 2. If UP# (I) is asserted, SRESET will not have an effect on the host microprocessor. For soft resets, SRESET should remain active for at least 15 CLK periods. SRESET is active HIGH. SRESET is asynchronous but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
SMI#	I	The **System Management Interrupt** input is used to invoke the System Management Mode (SMM). SMI# is a falling edge triggered signal which forces the CPU into SMM at the completion of the current instruction. SMI# is recognized on an instruction boundary and at each iteration for repeat string instructions. SMI# does not break LOCKed bus cycles and cannot interrupt a currently executing SMM. The CPU will latch the falling edge of one pending SMI# signal while the CPU is executing an existing SMI. The nested SMI will not be recognized until after the execution of a Resume (RSM) instruction.
SMIACT#	O	The **System Management Interrupt ACTive** is an active low output, indicating that the processor is operating in SMM. It is asserted when the CPU begins to execute the SMI state save sequence and will remain active LOW until the processor executes the last state restore cycle out of SMRAM.
STPCLK#	I	The **SToP CLocK request** input signal indicates a request has been made to turn off the CLK input. When the CPU recognizes a STPCLK#, the processor will stop execution on the next instruction boundary, unless superseded by a higher priority interrupt, empty all internal pipelines and the write buffers and generate a Stop Grant acknowledge bus cycle. STPCLK# is active LOW and is provided with an internal pull-up resistor. STPCLK# is an asynchronous signal, but must remain active until the processor issues the Stop Grant bus cycle. STPCLK# may be de-asserted at any time after the processor has issued the Stop Grant bus cycle. Note that STPCLK# should NOT be de-asserted before the CPU has issued the Stop Grant bus cycle.
UP#	I	The **Upgrade Present** input detects the presence of the upgrade processor, then powers down the core, and three-states all outputs of the original CPU, so that the original CPU consumes very low zero current. UP# is active LOW and sampled at all times, including after power-up and during reset.

Table 2-8. Output Pins

Name	Active Level	When Floated
BREQ	HIGH	
HLDA	HIGH	
BE3#–BE0#	LOW	Bus Hold
PWT, PCD	HIGH	Bus Hold
W/R#, M/IO#, D/C#	N/A	Bus Hold
LOCK#	LOW	Bus Hold
PLOCK#	LOW	Bus Hold
ADS#	LOW	Bus Hold
BLAST#	LOW	Bus Hold
PCHK#	LOW	
FERR#*	LOW	
A3–A2	N/A	Bus, Address Hold
SMIACT#	LOW	

Table 2-9. Input/Output Pins

Name	Active Level	When Floated
D31–D0	N/A	Bus Hold
DP3–DP0	HIGH	Bus Hold
A31–A4	N/A	Bus, Address Hold

NOTES:
All input/output signals are floated when UP# is asserted.
* Present on the Intel486 DX and DX2 CPUs only.

Table 2-10. Test Pins

Name	Input or Output	Sampled/Driven On
TCK	Input	N/A
TDI	Input	Rising Edge of TCK
TDO	Output	Falling Edge of TCK
TMS	Input	Rising Edge of TCK

Table 2-11. Input Pins

Name	Active Level	Synchronous/ Asynchronous	Internal Pull-up/ Pull-Down
CLK, CLK2			
RESET	HIGH	Asynchronous	
SRESET	HIGH	Asynchronous	Pull-down
HOLD	HIGH	Synchronous	
AHOLD	HIGH	Synchronous	Pull-down
EADS#	LOW	Synchronous	Pull-up
BOFF#	LOW	Synchronous	Pull-up
FLUSH#	LOW	Asynchronous	Pull-up
A20M#	LOW	Asynchronous	Pull-up
BS16#, BS8#	LOW	Synchronous	Pull-up
KEN#	LOW	Synchronous	Pull-up
RDY#	LOW	Synchronous	
BRDY#	LOW	Synchronous	Pull-up
INTR	HIGH	Asynchronous	
NMI	HIGH	Asynchronous	
IGNNE#*	LOW	Asynchronous	Pull-up
SMI#	LOW	Asynchronous	Pull-up
STPCLK#	LOW	Asynchronous	Pull-up
UP#	LOW		Pull-up
TCK	HIGH		Pull-up
TDI	HIGH		Pull-up
TMS	HIGH		Pull-up

* Present on the Intel486 DX and DX2 CPUs only.

3.0 CLOCK CONTROL

3.1 Clock Generation

The standard Intel486 SX and Intel486 DX CPUs are driven by a 1X clock as opposed to the Intel386 CPUs which use a 2X clock input. The SL Enhanced Intel486 CPUs are available with both the 1X and 2X clocking options. The 1X CLK input frequency is the same as the internal frequency of the CPU. The 1X clock allows simpler system design by cutting in half the clock speed required in the external system. The CLK2 input frequency for 2X CLK CPUs is twice the frequency of the CPU. **Clock control for CPUs with the 2X clock option is described in section 7.3. The remainder of Section 3 discusses clock control for 1X clock CPUs.**

The 1X clock relies on an internal Phase Lock Loop (PLL) to generate the two internal clock phases, "phase one" and "phase two". The rising edge of CLK corresponds to the start of phase one (ph1). All external timing parameters are specified with respect to the rising edge of CLK. The PLL requires a constant frequency CLK input (to within 0.1%), and therefore the CLK input cannot be changed dynamically.

On processors which use an internal clock doubling (DX2), CLK provides the fundamental timing reference for the bus interface unit. The CLK input is doubled internally so that the CPU core will operate at twice the CLK input frequency, hence twice the bus frequency. The internal clock doubler enhances all operations operating out of the internal cache and/or not blocked by external bus accesses. This mode also uses a Phase Lock Loop (PLL) and therefore the CLK input must be maintained at a constant frequency.

3.2 Stop Clock

The SL Enhanced Intel486 CPU provides an interrupt mechanism, STPCLK#, that allows system hardware to control the power consumption of the CPU by stopping the internal clock (output of the PLL) to the CPU core in a controlled manner. This low-power state is called the Stop Grant state. In addition, the STPCLK# interrupt allows the system to change the input frequency within the specified range or completely stop the CLK input frequency (input to the PLL). If the CLK input is completely stopped, the CPU enters into the Stop Clock state—the lowest power state.

STPCLK# is active LOW and is provided with an internal pull-up resistor. STPCLK# is an asynchronous signal, but must remain active until the processor issues the Stop Grant but cycle. STPCLK# may be de-asserted at any time after the processor has issued the Stop Grant bus cycle. Note that STPCLK# should NOT be de-asserted before the CPU has issued the Stop Grant bus cycle. When the CPU enters the Stop Grant state, the internal pull-up resistor is disabled so that the CPU power consumption is reduced. The STPCLK# input must be driven high (not floated) in order to exit the Stop Grant state. STPCLK# must be deasserted for a minimum of 5 clocks after RDY# or BRDY# is returned active for the Stop Grant bus cycle before being asserted again (see Figure 3-1).

There are two targets for the low-power mode supply current:

~20 mA–55 mA in the **Stop Grant state** (fast wake-up, frequency- and voltage-dependent).
and
~100 μA–200 μA in the full **Stop Clock state** (slow wake-up, voltage-dependent).

These are explained in further detail in Sections 3.5.2 and 3.5.3.

When the CPU recognizes a STPCLK# interrupt, the processor will stop execution on the next instruction boundary (unless superseded by a higher priority interrupt), stop the pre-fetch unit, empty all internal pipelines and the write buffers, generate a Stop Grant bus cycle, and then stop the internal clock. At this point the CPU is in the Stop Grant state.

The CPU cannot respond to a STPCLK# request from an HLDA state because it cannot empty the write buffers and, therefore, cannot generate a Stop Grant cycle.

The rising edge of STPCLK# will tell the CPU that it can return to program execution at the instruction following the interrupted instruction.

Unlike the normal interrupts, INTR and NMI, the STPCLK# interrupt does not initiate interrupt acknowledge cycles or interrupt table reads. Among external interrupts, STPCLK#'s order of priority is shown as follows:

External Events in Order of Priority

1) RESET/SRESET
2) FLUSH#
3) SMI#
4) NMI
5) INTR
6) STPCLK#

STPCLK# will be recognized while in an interrupt service routine or an SMM handler.

3.3 Stop Grant Bus Cycle

A special Stop Grant bus cycle will be driven to the bus after the CPU recognizes the STPCLK# interrupt. The definition of this bus cycle is the same as the HALT cycle definition for the standard Intel486 microprocessor, with the exception that the Stop Grant bus cycle drives the value 0000 0010H on the address pins. The system hardware must acknowledge this cycle by returning RDY# or BRDY#. **The CPU will not enter the Stop Grant state until either RDY# or BRDY# has been returned.**

The Stop Grant bus cycle is defined as follows:

M/IO# = 0, D/C# = 0, W/R# = 1, Address Bus = 0000 0010H (A_4 = 1), BE3#–BE0# = 1011, Data bus = undefined

The latency between a STPCLK# request and the Stop Grant bus cycle is dependent on the current instruction, the amount of data in the CPU write buffers, and the system memory performance.

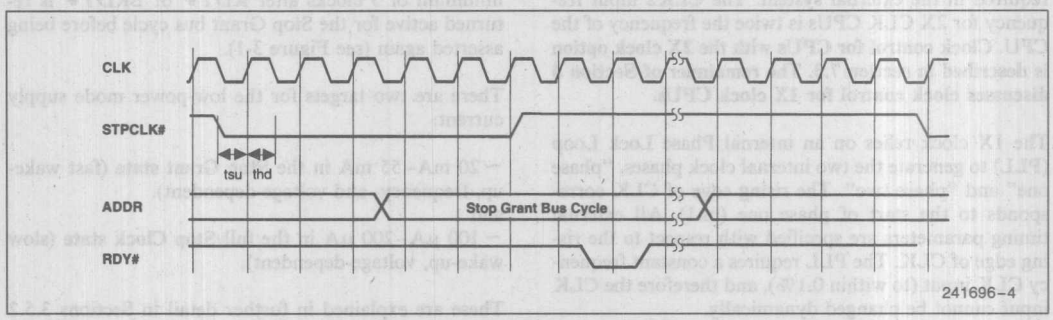


Figure 3-1. Stop Clock Protocol

3.4 Pin State During Stop Grant

During the Stop Grant state, most output and input/output signals of the microprocessor will maintain their previous condition (the level they held when entering the Stop Grant state). The data and data parity signals will be three-stated. In response to HOLD being driven active during the Stop Grant state (when the CLK input is running), the CPU will generate HLDA and three-state all output and input/output signals that are three-stated during the HOLD/HLDA state. After HOLD is de-asserted all signals will return to their prior state before the HOLD/HLDA sequence.

Table 3-1. Pin State during Stop Grant Bus State

Signal	Type	State
A3–A2	O	Previous State
A31–A4	I/O	Previous State
D31–D0	I/O	Floated
BE3#–BE0#	O	Previous State
DP3–DP0	I/O	Floated
W/R#, D/C#, M/IO#	O	Previous State
ADS#	O	Inactive
LOCK#, PLOCK#	O	Inactive
BREQ	O	Previous State
HLDA	O	As per HOLD
BLAST#	O	Previous State
FERR#	O	Previous State
PWT/PCD	O	Previous State
PCHK#	O	Previous State
SMIACT#	O	Previous State

In order to achieve the lowest possible power consumption during the Stop Grant state, the system designer must ensure the input signals with pull-up resistors are not driven LOW and the input signals with pull-down resistors are not driven HIGH. (See Table 2-10 in the Quick Pin Reference section for signals with internal pull-up and pull-down resistors).

All inputs, except data bus pins must be driven to the power supply rails to ensure the lowest possible current consumption during Stop Grant or Stop Clock modes. For compatibility with future processors, data pins must be driven low to achieve the lowest possible power consumption.

3.5 Clock Control State Diagram

The following state descriptions and diagram show the state transitions during a Stop Clock cycle for 1X clock mode (including the Intel486 DX2 CPU clock mode). Refer to Figure 3-2 for the Stop Clock state diagram.

3.5.1 NORMAL STATE

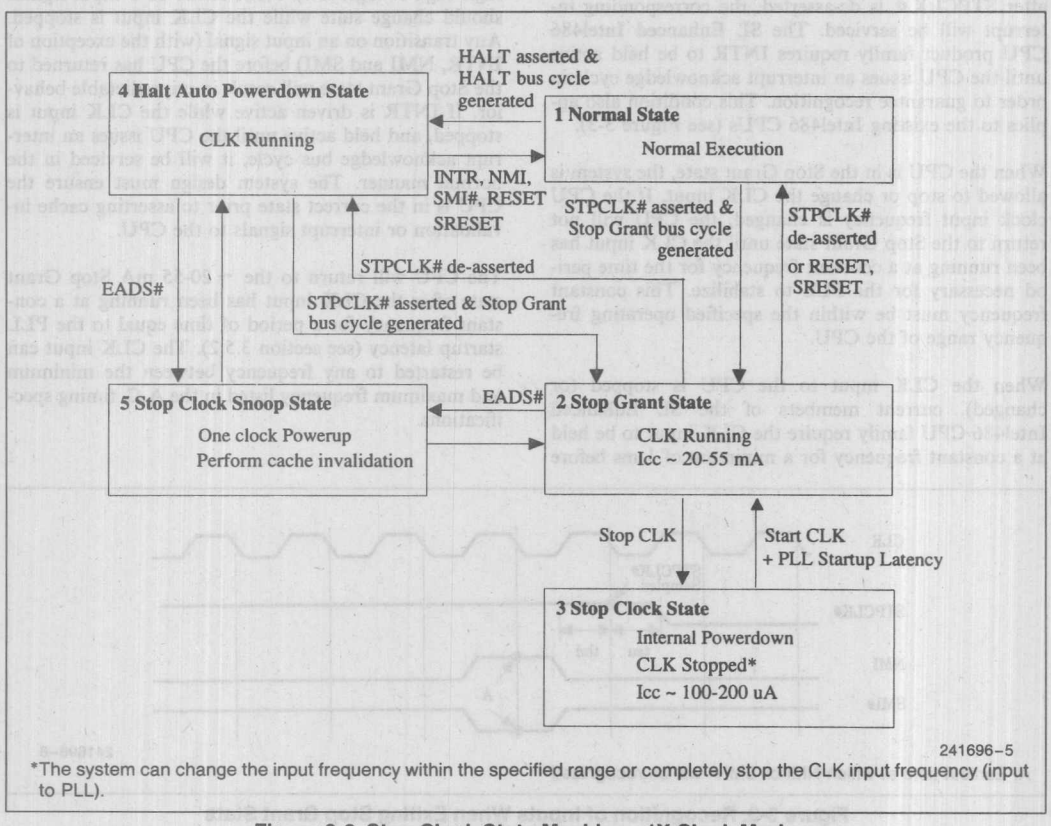
This is the normal operating state of the CPU.

3.5.2 STOP GRANT STATE

The **Stop Grant state** (~20–55 mA) provides a fast wake-up state that can be entered by simply asserting the external STPCLK# interrupt pin. Once the Stop Grant bus cycle has been placed on the bus, and either RDY# or BRDY# is returned, the CPU is in this ~20–55 mA state (depending on the CLK input frequency). The CPU returns to the normal execution state 10–20 clock periods after STPCLK# has been de-asserted.

While in the Stop Grant state, the pull-up resistors on STPCLK# and UP# are disabled internally. The system must continue to drive these inputs to the state

2



*The system can change the input frequency within the specified range or completely stop the CLK input frequency (input to PLL).

Figure 3-2. Stop Clock State Machine - 1X Clock Mode

they were in immediately before the CPU entered the Stop Grant state. For minimum CPU power consumption, all other input pins should be driven to their inactive level while the CPU is in the Stop Grant state.

A RESET or SRESET will bring the CPU from the Stop Grant state to the Normal state. The CPU will recognize the inputs required for cache invalidation's (HOLD, AHOLD, BOFF# and EADS#) as explained later in this section. The CPU will not recognize any other inputs while in the Stop Grant state. Input signals to the CPU will not be recognized until 1 CLK after STPCLK# is de-asserted (see Figure 3-3).

While in the Stop Grant state, the CPU will not recognize transitions on the interrupt signals (SMI#, NMI, and INTR). Driving an active edge on either SMI# or NMI will not guarantee recognition and service of the interrupt request following exit from the Stop Grant state. However, if one of the interrupt signals (SMI#, NMI, or INTR) is driven active while the CPU is in the Stop Grant state, and held active for at least one CLK after STPCLK# is de-asserted, the corresponding interrupt will be serviced. The SL Enhanced Intel486 CPU product family requires INTR to be held active until the CPU issues an interrupt acknowledge cycle in order to guarantee recognition. This condition also applies to the existing Intel486 CPUs (see Figure 3-3).

When the CPU is in the Stop Grant state, the system is allowed to stop or change the CLK input. If the CPU clock input frequency is changed, the CPU will not return to the Stop Grant state until the CLK input has been running at a constant frequency for the time period necessary for the PLL to stabilize. This constant frequency must be within the specified operating frequency range of the CPU.

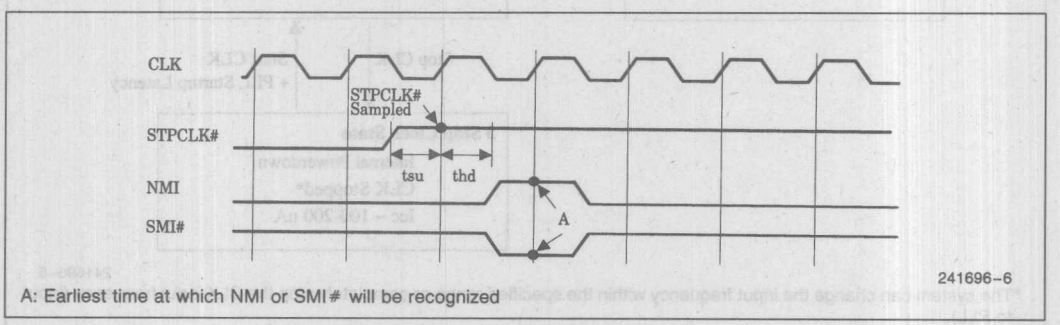
When the CLK input to the CPU is stopped (or changed), current members of the SL Enhanced Intel486 CPU family require the CLK input to be held at a constant frequency for a minimum of 1 ms before de-asserting STPCLK#. This 1 ms time period is necessary so that the PLL can stabilize, and it must be met before the CPU will return to the Stop Grant state. Future versions of the SL Enhanced Intel486 CPU family may require substantially less than 1 ms for the PLL to stabilize. In order to take advantage of a shorter PLL stabilization period, a system design should provide for a programmable time period between restarting the CPU clock and de-asserting STPCLK#.

The CPU will generate a Stop Grant bus cycle only when entering that state from the Normal or the Auto HALT Power Down state. When the CPU enters the Stop Grant state from the Stop Clock state or the Stop Clock Snoop state, the CPU will not generate a Stop Grant bus cycle.

3.5.3 STOP CLOCK STATE

Stop Clock state (~100–200 μA) is entered from the Stop Grant state by stopping the CLK input (either logic high or logic low). None of the CPU input signals should change state while the CLK input is stopped. Any transition on an input signal (with the exception of INTR, NMI and SMI) before the CPU has returned to the Stop Grant state will result in unpredictable behavior. If INTR is driven active while the CLK input is stopped, and held active until the CPU issues an interrupt acknowledge bus cycle, it will be serviced in the normal manner. The system design must ensure the CPU is in the correct state prior to asserting cache invalidation or interrupt signals to the CPU.

The CPU will return to the ~20-55 mA Stop Grant state after the CLK input has been running at a constant frequency for a period of time equal to the PLL startup latency (see section 3.5.2). The CLK input can be restarted to any frequency between the minimum and maximum frequency listed in the A.C. timing specifications.



A: Earliest time at which NMI or SMI# will be recognized

Figure 3-3. Recognition of Inputs When Exiting Stop Grant State

3.5.4 AUTO HALT POWER DOWN STATE

The execution of a HALT instruction will also cause the CPU to automatically enter a ~20–55 mA state, called the **Auto HALT Power Down state**. The CPU will issue a normal HALT bus cycle when entering this state. The CPU will transition to the Normal state on the occurrence of INTR, NMI, SMI#, RESET, or SRESET.

The system can generate a STPCLK# while the CPU is in the Auto HALT Power Down state. The CPU will generate a Stop Grant bus cycle when it enters the Stop Grant State from the HALT state. When the system de-asserts the STPCLK# interrupt, the CPU will return execution to the HALT state. The CPU will generate a new HALT bus cycle when it re-enters the HALT state from the Stop Grant state.

3.5.5 STOP CLOCK SNOOP STATE (CACHE INVALIDATIONS)

When the CPU is in the Stop Grant state or the Auto HALT Power Down state, the CPU will recognize HOLD, AHOLD, BOFF# and EADS# for cache invalidation. When the system asserts HOLD, AHOLD, or BOFF#, the CPU will float the bus accordingly. When the system then asserts EADS#, the CPU will transparently enter the **Stop Clock Snoop state** and will power up for 1 full core clock in order to perform the required cache snoop cycle. It will then re-freeze the clock to the CPU core and return to the previous state. The CPU does not generate a bus cycle when it returns to the previous state.

A FLUSH# event during the Stop Grant state or the Auto HALT Power Down state will be latched and acted upon by asserting the internal FLUSH# signal for one clock upon re-entering the Normal state.

3.5.6 AUTO IDLE POWER DOWN STATE

When the chip is known to be truly idle and waiting for a ready from a memory or I/O bus cycle read, the Intel486 DX2 CPU will reduce its core clock rate to half of its original frequency without affecting performance. When any ready is asserted, the part will return to clocking the core at the doubled DX2 clock. This functionality is transparent to software and external hardware. This feature applies only to those members of the SL Enhanced Intel486 CPU family that have a doubled core clock rate. Auto Idle Power Down provides an average power savings of 10%.

3.6 Supply Current Model for Stop Clock Modes and Transitions

The following diagram illustrates the effect of different Stop Clock state transitions on the supply current.

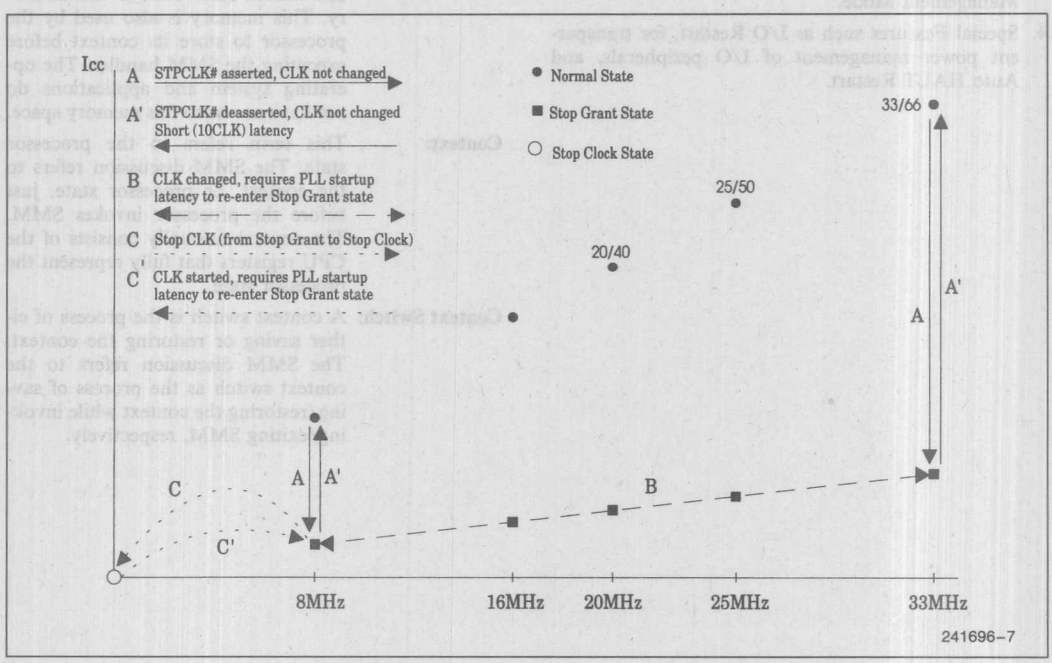


Figure 3-4. Supply Current Model for Stop Clock Modes and Transitions

4.0 INTEL'S SYSTEM MANAGEMENT MODE ARCHITECTURE

4.1 SMM Overview

The SL Enhanced Intel486 microprocessor supports four modes: **Real, Virtual-86, Protected and System Management Mode (SMM)**. As an operating mode, SMM has its distinct processor environment, interface and hardware/software features.

SMM provides the system designer with a means of adding new software controlled features to their computer products which always operate transparent to the Operating System (OS) and software applications. SMM is intended for use only by system firmware, not by applications software or general purpose systems software.

The SMM architectural extension consists of the following elements:

1. A System Management Interrupt (SMI#) hardware interface.
2. A dedicated and secure memory space (SMRAM) for SMI handler code and CPU state (context) data with a status signal for the system to decode access to that memory space, SMIACT#.
3. Resume (RSM) instruction, for exiting the System Management Mode.
4. Special Features such as I/O Restart, for transparent power management of I/O peripherals, and Auto HALT Restart.

4.2 Terminology

The following terms are used throughout the discussion of System Management Mode.

SMM: System Management Mode. This is the operating environment that the processor (system) enters when the System Management Interrupt is being serviced.

SMI: System Management Interrupt. This is part of the SMM interface. When SMI# is asserted (SMI# pin asserted low) it causes the processor to invoke SMM. **The SMI# pin is the only means of entering SMM.**

SMM handler: System Management Mode handler. This is the code that will be executed when the processor is in SMM. An example application that this code might implement is a power management control or a system control function.

RSM: Resume instruction. This instruction is used by the SMM handler to exit the SMM and return to the interrupted OS or Application process.

SMRAM: This is the physical memory dedicated to SMM. The SMM handler code and related data reside in this memory. This memory is also used by the processor to store its context before executing the SMM handler. The operating system and applications do not have access to this memory space.

Context: This term refers to the processor state. The SMM discussion refers to the context, or processor state, just before the processor invokes SMM. The context normally consists of the CPU registers that fully represent the processor state.

Context Switch: A context switch is the process of either saving or restoring the context. The SMM discussion refers to the context switch as the process of saving/restoring the context while invoking/exiting SMM, respectively.

4.3 System Management Interrupt Processing

The system interrupts the normal program execution and invokes SMM by generating a System Management Interrupt (SMI#) to the CPU. The CPU will service the SMI# by executing the following sequence.

1. The CPU asserts the SMIACT# signal, indicating to the system that it should enable the SMRAM.
2. The CPU saves its state (context) to SMRAM, starting at address location 3FFFFH, proceeding downward in a stack-like fashion.
3. The CPU switches to the System Management Mode processor environment (a pseudo-real mode).
4. The CPU will then jump to the absolute address of 38000H in SMRAM to execute the SMI handler. This SMI handler performs the system management activities.
5. The SMI handler will then execute the RSM instruction which restores the CPU's context from SMRAM, de-asserts the SMIACT# signal, and then returns control to the previously interrupted program execution.

The System Management Interrupt hardware interface consists of the SMI# interrupt request input and the SMIACT# output used by the system to decode the SMRAM.

4.3.1 SYSTEM MANAGEMENT INTERRUPT (SMI#)

SMI# is a falling-edge triggered, non-maskable interrupt request signal. SMI# is an asynchronous signal, but setup and hold times, t20 and t21, must be met in order to guarantee recognition in a specific clock. The SMI# input need not remain active until the interrupt is actually serviced. The SMI# input only needs to remain active for a single clock if the required setup and hold times are met. SMI# will also work correctly if it is held active for an arbitrary number of clocks.

The SMI# input must be held inactive for at least four clocks after it is asserted to reset the edge triggered logic.

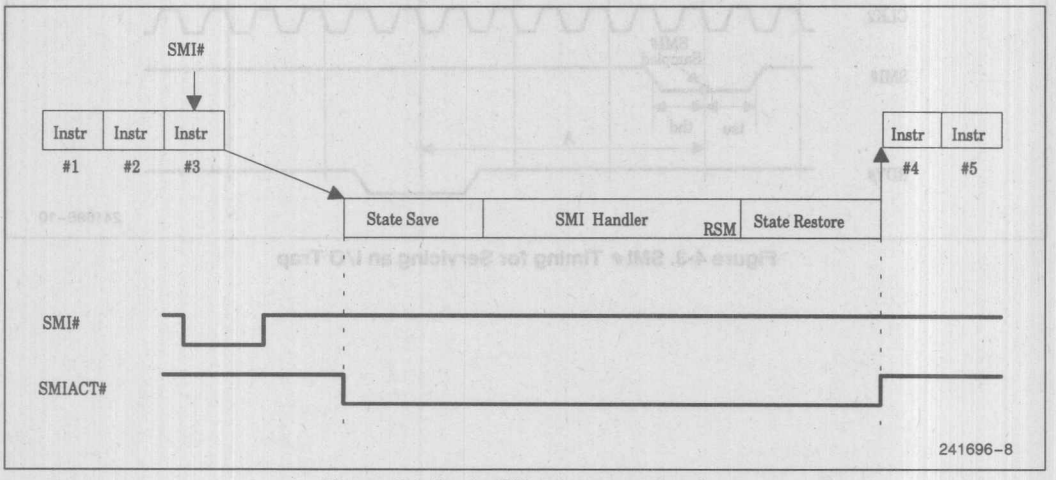


Figure 4-1. Basic SMI# Interrupt Service

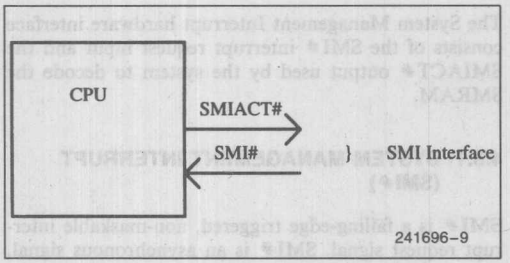


Figure 4-2. Basic SMI# Hardware Interface

SMI#, like NMI, is not affected by the IF bit in the EFLAGS register and is recognized on an instruction boundary. An SMI# will not break locked bus cycles. The SMI# has a higher priority than NMI and is not masked during an NMI.

After the SMI# interrupt is recognized, the SMI# signal will be masked internally until the RSM instruction is executed and the interrupt service routine is complete. Masking the SMI# prevents recursive SMI# calls. If another SMI# occurs while the SMI# is masked, the pending SMI# will be recognized and executed on the next instruction boundary after the current SMI# completes. This instruction boundary occurs before execution of the next instruction in the interrupted application code, resulting in back to back SMM handlers. Only one SMI# can be pending while SMI# is masked.

The SMI# signal is synchronized internally and must be asserted at least three (3) CLK periods prior to asserting the RDY# signal in order to guarantee recognition on a specific instruction boundary. This is important for servicing an I/O trap with an SMI# handler. In order for SMI# to be recognized with respect to SRESET, SMI# should not be asserted until two (2) clocks after SRESET becomes inactive.

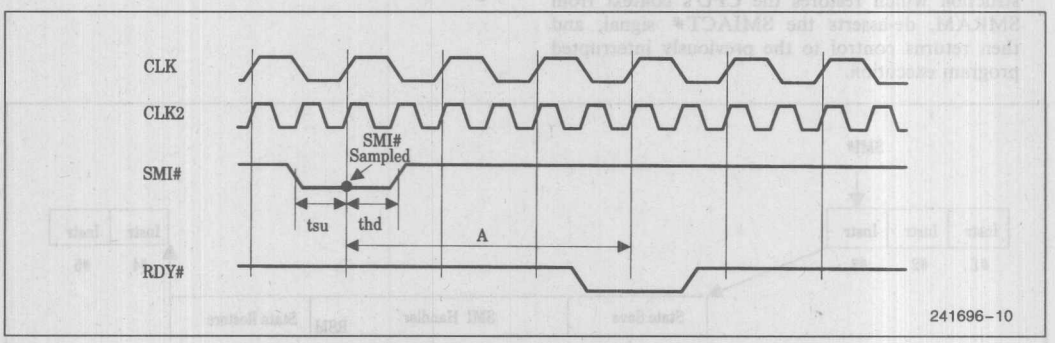


Figure 4-3. SMI# Timing for Servicing an I/O Trap

4.3.2 SMI ACTIVE (SMIACT#)

SMIACT# indicates that the CPU is operating in System Management Mode. The CPU asserts SMIACT# in response to an SMI interrupt request on the SMI# pin. SMIACT# is driven active after the CPU has completed all pending write cycles (including emptying the write buffers), and before the first access to SMRAM when the CPU saves (writes) its state (or context) to SMRAM. SMIACT# remains active until the last access to SMRAM when the CPU restores (reads) its state from SMRAM. The SMIACT# signal does not float in response to HOLD. The SMIACT# signal is used by the system logic to decode SMRAM.

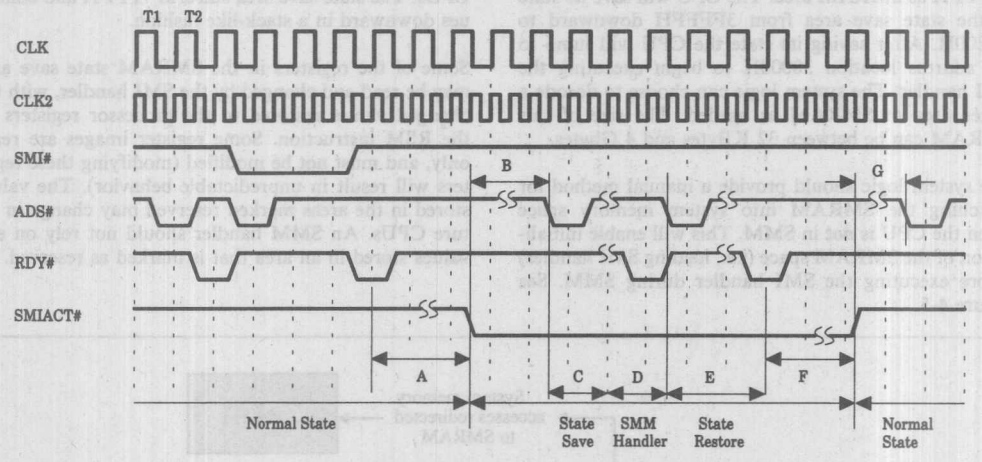
NOTE:
The number of CLKs required to complete the SMM state save and restore is very dependent on system memory performance. The values shown in Figure 4-4 assume 0 wait-state memory writes (2 CLK cycles), 2-1-1-1 burst read cycles, and 0 wait-state non-burst reads (2 CLK cycles). Additionally, it is assumed that the data read during the SMM state restore sequence is not cacheable.

As shown in Figure 4-4, the minimum time required to enter an SMI handler routine for the Intel486 DX CPU (from the completion of the interrupted instruction) is given by:

Latency to beginning of SMI handler = A + B + C = 306 CLKs

and the minimum time required to return to the interrupted application (following the final SMM instruction before RSM) is given by:

Latency to continue interrupted application = E + F + G = 327 CLKs



	Intel486 DX CPU	Intel486 DX2 CPU
A: Last RDY# from non-SMM transfer to SMIACT# assertion	2 CLK minimum	2 CLK minimum
B: SMIACT# assertion to first ADS# for SMM state save	40 CLK minimum	20 CLK minimum
C: SMM state save (dependent on memory performance)	Approximately 264 CLKs	Approximately 139 CLKs
D: SMM handler	User determined	User determined
E: SMM state restore (dependent on memory performance)	Approximately 303 CLKs	Approximately 236 CLKs
F: Last RDY# from SMM transfer to de-assertion of SMIACT#	4 CLK minimum	2 CLK minimum
G: SMIACT# de-assertion to first non-SMM ADS#	20 CLK minimum	20 CLK minimum

Figure 4-4. SMIACT# Timing

4.3.3 SMRAM

The CPU uses the SMRAM space for state save and state restore operations during an SMI. The SMI handler, which also resides in SMRAM, uses the SMRAM space to store code, data and stacks. In addition, the SMI handler can use the SMRAM for system management information such as the system configuration, configuration of a powered-down device, and system designer-specific information.

The CPU asserts the SMIACT# output to indicate to the memory controller that it is operating in System Management Mode. The system logic should ensure that only the CPU has access to this area. Alternate bus masters or DMA devices trying to access the SMRAM space when SMIACT# is active should be directed to system RAM in the respective area.

The system logic is minimally required to decode the physical memory address range from 38000H–3FFFFH as SMRAM area. The CPU will save its state to the state save area from 3FFFFH downward to 3FE00H. After saving its state the CPU will jump to the address location 38000H to begin executing the SMI handler. The system logic can choose to decode a larger area of SMRAM as needed. The size of this SMRAM can be between 32 KBytes and 4 Gbytes.

The system logic should provide a manual method for switching the SMRAM into system memory space when the CPU is **not** in SMM. This will enable initialization of the SMRAM space (i.e., loading SMI handler) before executing the SMI handler during SMM. See Figure 4-5.

4.3.3.1 SMRAM State Save Map

When the SMI# is recognized on an instruction boundary, the CPU core first sets the SMIACT# signal LOW indicating to the system logic that accesses are now being made to the system-defined SMRAM areas. The CPU then writes its state to the state save area in the SMRAM. The state save area starts at CS Base + [8000H + 7FFFH]. The default CS Base is 30000H, therefore the default state save area is at 3FFFFH. In this case, the CS Base can also be referred to as the SMBASE.

If the *SMBASE Relocation feature* is enabled, then the SMRAM addresses can change. The following formula is used to determine the relocated addresses where the context is saved. The context will reside at CS Base + [8000H + Register Offset], where the default initial CS Base is 30000H and the Register Offset is listed in the SMRAM state save map (Table 4-1). Reserved spaces will be used to accommodate new registers in future CPUs. The state save area starts at 7FFFH and continues downward in a stack-like fashion.

Some of the registers in the SMRAM state save area may be read and changed by the SMI handler, with the changed values restored to the processor registers by the RSM instruction. Some register images are read-only, and must not be modified (modifying these registers will result in unpredictable behavior). The values stored in the areas marked reserved may change in future CPUs. An SMM handler should not rely on any values stored in an area that is marked as reserved.

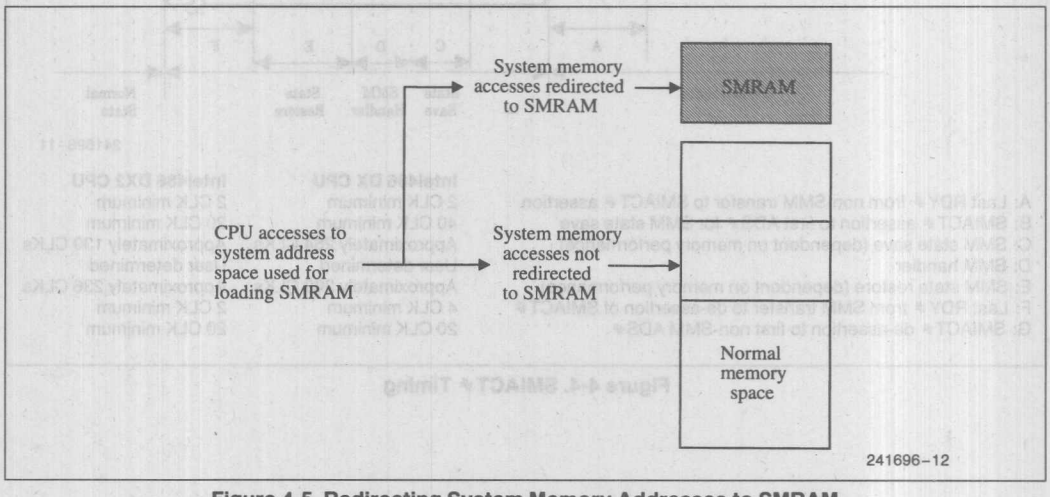


Figure 4-5. Redirecting System Memory Addresses to SMRAM

Table 4-1. SMRAM State Save Map

Register Offset	Register	Writeable?
7FFC	CR0	NO
7FF8	CR3	NO
7FF4	EFLAGS	YES
7FF0	EIP	YES
7FEC	EDI	YES
7EE8	ESI	YES
7FE4	EBP	YES
7FE0	ESP	YES
7FDC	EBX	YES
7FD8	EDX	YES
7FD4	ECX	YES
7FD0	EAX	YES
7FCC	DR6	NO
7FC8	DR7	NO
7FC4	TR*	NO
7FC0	LDTR*	NO
7FBC	GS*	NO
7FB8	FS*	NO
7FB4	DS*	NO
7FB0	SS*	NO
7FAC	CS*	NO
7FA8	ES*	NO
7FA7–7F98	Reserved	NO
7F94	IDT Base	NO
7F93–7F8C	Reserved	NO
7F88	GDT Base	NO
7F87–7F04	Reserved	NO
7F02	Auto HALT Restart Slot (Word)	YES
7F00	I/O Trap Restart Slot (Word)	YES
7EFC	SMM Revision Identifier (Dword)	NO
7EF8	SMBASE Slot (Dword)	YES
7EF7–7E00	Reserved	NO

NOTES:
*Upper two bytes are reserved
Modifying a value that is marked as not writeable will result in unpredictable behavior.
Words are stored in 2 consecutive bytes in memory with the low-order byte at the lowest address and the high-order byte in the high address.

The following registers are saved and restored (in areas of the state save that are marked reserved) but are not visible to the system software programmer:

CR1, CR2 and CR4, hidden descriptor registers for CS, DS, ES, FS, GS.

If an SMI request is issued for the purpose of powering down the CPU, the values of all reserved locations in the SMM state save must be saved to non-volatile memory.

The following registers are not automatically saved and restored by SMI# and RSM:

DR5–DR0, TR7–TR3, FPU registers: STn, FCS, FSW, tag word, FP instruction pointer, FP op code, and operand pointer.

For all SMI requests except for suspend/resume, these registers do not have to be saved as their contents will not change. During a power down suspend/resume however, a resume reset will clear these registers back to their default values. In this case, the suspend SMI handler should read these registers directly to save them and restore them during the power up resume. Anytime the SMI handler changes these registers in the CPU, it must also save and restore them.

4.3.4 EXIT FROM SMM

The **RSM** instruction, is only available to the SMI handler. The op code of the instruction is 0FAAH. Execution of this instruction while the CPU is executing outside of SMM will cause an invalid op-code error. The last instruction of the SMI handler will be the RSM instruction.

The RSM instruction restores the state save image from SMRAM back to the CPU, then returns control back to the interrupted program execution. There are three SMM features that can be enabled by writing to control "slots" in the SMRAM state save area.

Auto HALT Restart. It is possible for the SMI request to interrupt the HALT state. The SMI handler can tell the RSM instruction to return control to the HALT instruction **or** to return control to the instruction following the HALT instruction by appropriately setting the Auto HALT Restart slot. The default operation is to restart the HALT instruction.

I/O Trap Restart. If the SMI# interrupt was generated on an I/O access to a powered-down device, the SMI handler can tell the RSM instruction to re-execute that I/O instruction by setting the I/O Trap Restart slot.

SMBASE Relocation. The system can relocate the SMRAM by setting the SMBASE Relocation slot in the state save area. The RSM instruction will set the SMBASE in the processor based on the value in the SMBASE relocation slot. The SMBASE must be 32K aligned.

For further details on these SMM features, see Section 4.5.

If the processor detects invalid state information, it enters the shutdown state; this happens only in the following situations:

- The value stored in the SMBASE slot is not a 32 Kbyte-aligned address.
- A reserved bit of CR4 is set to 1.
- A combination of bits in CR0 is illegal; namely, (PG=1 and PE=0) or (NW=1 and CD=0).

In shutdown mode, the processor stops executing instructions until an NMI interrupt is received or reset initialization is invoked. The processor generates a special bus cycle to indicate it has entered shutdown mode.

4.4 System Management Mode Programming Model

4.4.1 ENTERING SYSTEM MANAGEMENT MODE

SMM is one of the major operating modes, on a level with Protected mode, Real address mode or virtual-86 mode. Figure 4-6 shows how the processor can enter SMM from any of the three modes and then return.

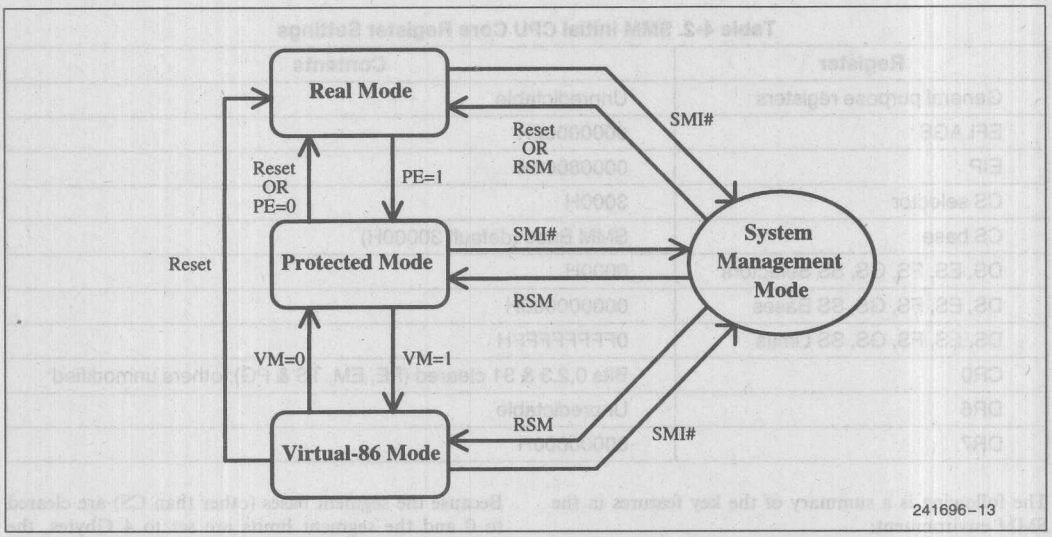


Figure 4-6. Transition to and from System Management Mode

The external signal SMI# causes the processor to switch to SMM. The RSM instruction exits SMM. SMM is transparent to applications programs and operating systems because of the following:

- The only way to enter SMM is via a type of non-maskable interrupt triggered by an external signal.
- The processor begins executing SMM code from a separate address space, referred to earlier as system management RAM (SMRAM).
- Upon entry into SMM, the processor saves the register state of the interrupted program in a part of SMRAM called the SMM context save space.
- All interrupts normally handled by the operating system or by applications are disabled upon entry into SMM.
- A special instruction, RSM, restores processor registers from the SMM context save space and returns control to the interrupted program.

SMM is similar to Real address mode in that there are no privilege levels or address mapping. An SMM program can execute all I/O and other system instructions and can address up to four Gbytes of memory.

4.4.2 PROCESSOR ENVIRONMENT

When an SMI# signal is recognized on an instruction execution boundary, the processor waits for all stores to complete, including emptying of the write buffers. The final write cycle is complete when the system returns RDY# or BRDY#. The processor then drives SMIACT# active, saves its register state to SMRAM space, and begins to execute the SMM handler.

SMI# has greater priority than debug exceptions and external interrupts. This means that if more than one of these conditions occur at an instruction boundary, only the SMI# processing occurs, not a debug exception or external interrupt. Subsequent SMI# requests are not acknowledged while the processor is in SMM. The first SMI# interrupt request that occurs while the processor is in SMM is latched, and serviced when the processor exits SMM with the RSM instruction. Only one SMI# will be latched by the CPU while it is in SMM.

When the CPU invokes SMM, the CPU core registers are initialized as follows:

Table 4-2. SMM Initial CPU Core Register Settings

Register	Contents
General purpose registers	Unpredictable
EFLAGS	00000002H
EIP	00008000H
CS selector	3000H
CS base	SMM Base (default 30000H)
DS, ES, FS, GS, SS Selectors	0000H
DS, ES, FS, GS, SS Bases	000000000H
DS, ES, FS, GS, SS Limits	0FFFFFFFFH
CR0	Bits 0,2,3 & 31 cleared (PE, EM, TS & PG); others unmodified
DR6	Unpredictable
DR7	00000000H

The following is a summary of the key features in the SMM environment:

1. Real mode style address calculation
2. Gbyte limit checking
3. IF flag is cleared
4. NMI is disabled
5. TF flag in EFLAGS is cleared; single step traps are disabled
6. DR7 is cleared; debug traps are disabled
7. The RSM instruction no longer generates an invalid op code error
8. Default 16-bit op code, register and stack use.

All bus arbitration (HOLD, AHOLD, BOFF#) inputs and bus sizing (BS8#, BS16#) inputs operate normally while the CPU is in SMM.

4.4.3 EXECUTING SYSTEM MANAGEMENT MODE HANDLER

The processor begins execution of the SMM handler at offset 8000H in the CS segment. The CS Base is initially 30000H. The CS Base can be changed, however, *using the SMM Base relocation feature.*

When the SMM handler is invoked, the CPU's PE and PG bits in CR0 are reset to 0. The processor is in an environment similar to Real mode, but without the 64 Kbyte limit checking. However, the default operand size and the default address size are set to 16 bits.

The EM bit is cleared so that no exceptions are generated. (If the SMM was entered from Protected mode, the Real mode interrupt and exception support is not available.) The SMI handler should not use floating point unit instructions until the FPU is properly detected (within the SMI handler) and the exception support is initialized.

Because the segment bases (other than CS) are cleared to 0 and the segment limits are set to 4 Gbytes, the address space may be treated as a single flat 4 Gbyte linear space that is unsegmented. The CPU is still in Real mode and when a segment selector is loaded with a 16-bit value, that value is then shifted left by 4 bits and loaded into the segment base cache. The limits and attributes are not modified.

In SMM, the CPU can access or jump anywhere within the 4 Gbyte logical address space. The CPU can also indirectly access or perform a near jump anywhere within the 4 Gbyte logical address space.

4.4.3.1 Exceptions and Interrupts within System Management Mode

When the CPU enters SMM, it disables INTR interrupts, debug and single step traps by clearing the EFLAGS, DR6 and DR7 registers. This is done to prevent a debug application from accidentally breaking into an SMM handler. This is necessary because the SMM handler operates from a distinct address space (SMRAM) and hence the debug trap will not represent the normal system memory space.

If an SMM handler wishes to use the debug trap feature of the processor to debug SMM handler code, it must first ensure that an SMM compliant debug handler is available. The SMM handler must also ensure DR0–DR3 is saved to be restored later. The debug registers DR0–DR3 and DR7 must then be initialized with the appropriate values.

If the processor wishes to use the single step feature of the processor, it must ensure that an SMM compliant single step handler is available and then set the trap flag in the EFLAGS register.

If the system design requires the processor to respond to hardware INTR requests while in SMM, it must ensure that an SMM compliant interrupt handler is available and then set the interrupt flag in the EFLAGS register (using the STI instruction). Software interrupts are not blocked upon entry to SMM, and the system software designer must provide an SMM compliant interrupt handler before attempting to execute any software interrupt instructions. Note that in SMM mode the interrupt vector table has the same properties and location as the Real mode vector table.

NMI interrupts are blocked upon entry to the SMM handler. If an NMI request occurs during the SMM handler, it is latched and serviced after the processor exits SMM. Only one NMI request will be latched during the SMM handler. If an NMI request is pending when the processor executes the RSM instruction, the NMI is serviced before the next instruction of the interrupted code sequence.

Although NMI requests are blocked when the CPU enters SMM, they may be enabled through software by executing an IRET instruction. If the SMM handler requires the use of NMI interrupts, it should invoke a dummy interrupt service routine for the purpose of executing an IRET instruction. Once an IRET instruction is executed, NMI interrupt requests are serviced in the same "real mode" manner in which they are handled outside of SMM.

4.5 SMM Features

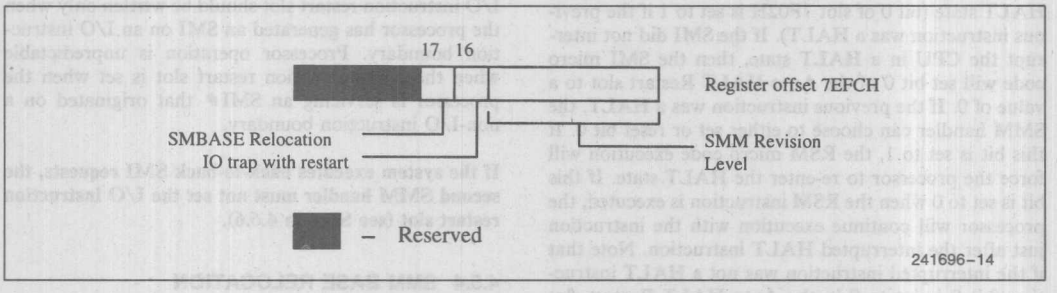
4.5.1 SMM REVISION IDENTIFIER

The SMM revision identifier is used to indicate the version of SMM and the SMM extensions that are supported by the processor. The SMM revision identifier is written during SMM entry and can be examined in SMRAM space at Register Offset 7EFCH. The lower word of the SMM revision identifier refers to the version of the base SMM architecture. The upper word of the SMM revision identifier refers to the extensions available.

Bit 16 of the SMM revision identifier is used to indicate to the SMM handler that this processor supports the SMM I/O trap extension. If this bit is high, then this processor supports the SMM I/O trap extension. If this bit is low, then this processor does not support I/O trapping using the I/O trap slot mechanism.

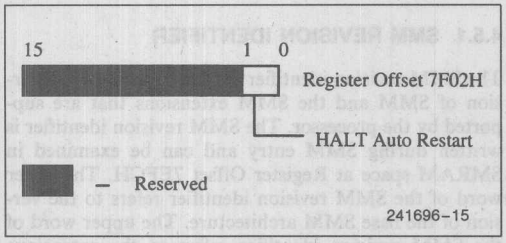
Bit 17 of this slot indicates whether the processor supports relocation of the SMM jump vector and the SMRAM base address.

All members of the SL Enhanced Intel486 CPU family support both the I/O Trap Restart and the SMBASE relocation features.



Bits	Value	Comments
16	0	Processor does not support I/O Trap Restart
16	1	Processor supports I/O Trap Restart
17	0	Processor does not support SMBASE Relocation
17	1	Processor supports SMBASE Relocation

4.5.2 AUTO HALT RESTART

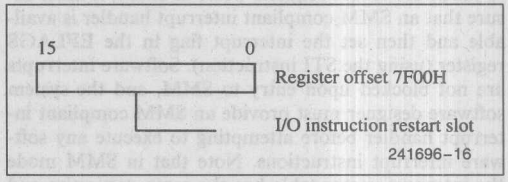


Value of Bit 0 at Entry	Value of Bit 0 at Exit	Comments
0	0	Returns to next instruction in interrupted program.
0	1	Unpredictable
1	0	Returns to next instruction after HALT
1	1	Returns to HALT state

The Auto HALT Restart slot at register offset (word location) 7F02H in SMRAM indicates to the SMM handler that the SMI interrupted the CPU during a HALT state (bit 0 of slot 7F02H is set to 1 if the previous instruction was a HALT). If the SMI did not interrupt the CPU in a HALT state, then the SMI micro code will set bit 0 of the Auto HALT Restart slot to a value of 0. If the previous instruction was a HALT, the SMM handler can choose to either set or reset bit 0. If this bit is set to 1, the RSM micro code execution will force the processor to re-enter the HALT state. If this bit is set to 0 when the RSM instruction is executed, the processor will continue execution with the instruction just after the interrupted HALT instruction. Note that if the interrupted instruction was not a HALT instruction (bit 0 is set to 0 in the Auto HALT Restart slot upon SMM entry), setting bit 0 to 1 will cause unpredictable behavior when the RSM instruction is executed.

If the HALT instruction is restarted, the CPU will generate a memory access to fetch the HALT instruction (if it is not in the internal cache), and execute a HALT bus cycle.

4.5.3 I/O INSTRUCTION RESTART



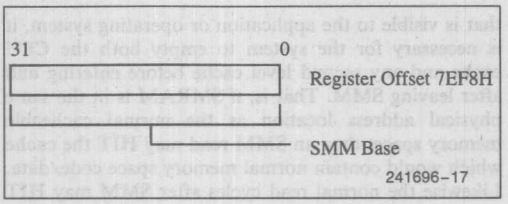
Value at Entry	Value at Exit	Comments
00H	00H	Do not restart trapped I/O instruction
00H	0FFH	Restart trapped I/O instruction

The I/O instruction restart slot (register offset 7F00H in SMRAM) gives the SMM handler the option of causing the RSM instruction to automatically re-execute the interrupted I/O instruction. When the RSM instruction is executed, if the I/O instruction restart slot contains the value 0FFH, then the CPU will automatically re-execute the I/O instruction that the SMI trapped. If the I/O instruction restart slot contains the value 00H when the RSM instruction is executed, then the CPU will not re-execute the I/O instruction. The CPU automatically initializes the I/O instruction restart slot to 00H during SMM entry. The I/O instruction restart slot should be written only when the processor has generated an SMI on an I/O instruction boundary. Processor operation is unpredictable when the I/O instruction restart slot is set when the processor is servicing an SMI# that originated on a non-I/O instruction boundary.

If the system executes back-to-back SMI requests, the second SMM handler must not set the I/O instruction restart slot (see Section 4.6.6).

4.5.4 SMM BASE RELOCATION

The SL Enhanced Intel486 CPU family provides a new control register, SMBASE. The address space used as SMRAM can be modified by changing the SMBASE register before exiting an SMI handler routine. SMBASE can be changed to any 32K aligned value (values that are not 32K aligned will cause the CPU to enter the shutdown state when executing the RSM instruction). SMBASE is set to the default value of 30000H on RESET, but is not changed on SRESET. If the SMBASE register is changed during an SMM handler, all following SMI# requests will initiate a state save at the new SMBASE.



The SMBASE slot in the SMM state save area is a feature used to indicate and change the SMI jump vector location and the SMRAM save area. When bit 17 of the SMM Revision Identifier is set then this feature exists and the SMRAM base and consequently the jump vector are as indicated by the SMM Base slot. During the execution of the RSM instruction, the CPU will read this slot and initialize the CPU to use the new SMBASE during the next SMI. During an SMI, the CPU will do its context save to the new SMRAM area pointed to by the SMBASE, store the current SMBASE in the SMM Base slot (offset 7EF8H), and then start execution of the new jump vector based on the current SMBASE.

The SMBASE must be a 32 Kbyte aligned, 32-bit integer that indicates a base address for the SMRAM context save area and the SMI jump vector. For example when the processor first powers up, the minimum SMRAM area is from 38000H–3FFFFH. The default SMBASE is 30000H. Hence the starting address of the jump vector is calculated by:

SMBASE + 8000H

While the starting address for the SMRAM state save area is calculated by:

SMM Base + [8000H + 7FFFH]

Hence when this feature is enabled the SMRAM register map is addressed according to the above formulae.

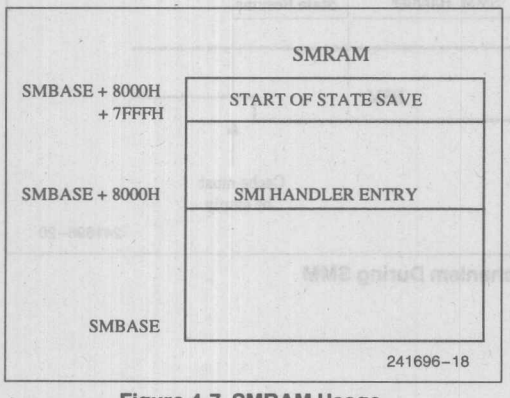


Figure 4-7. SMRAM Usage

To change the SMRAM base address and SMI jump vector location, the SMM handler should modify the SMBASE slot. Upon executing an RSM instruction, the processor will read the SMBASE slot and store it internally. Upon recognition of the next SMI request, the processor will use the new SMBASE slot for the SMRAM dump and SMI jump vector.

If the modified SMBASE slot does not contain a 32 KByte-aligned value, the RSM micro code will cause the CPU to enter the shutdown state.

4.6 SMM - System Design Considerations

4.6.1 SMRAM INTERFACE

The hardware designed to control the SMRAM space must follow these guidelines:

1. A provision should be made to allow for initialization of SMRAM space during system boot up. This initialization of SMRAM space must happen before the first occurrence of an SMI# interrupt. Initializing the SMRAM space must include installation of an SMM handler, and may include installation of related data structures necessary for particular SMM applications. The memory controller providing the interface to the SMRAM should provide a means for the initialization code to manually open the SMRAM space.
2. A minimum initial SMRAM address space of 38000H-3FFFFH should be decoded by the memory controller.
3. Alternate bus masters (such as DMA controllers) should not be allowed to access SMRAM space. Only the CPU, either through SMI or during initialization, should be allowed access to SMRAM.
4. In order to implement a zero-volt suspend function, the system must have access to all of normal system memory from within an SMM handler routine. If the SMRAM is going to overlay normal system memory, there must be a method of accessing any system memory that is located underneath SMRAM (see Section A.2.3).

There are two potential schemes for locating the SMRAM, either overlaid to an address space on top of normal system memory, or placed in a distinct address space. See Figure 4-8. When SMRAM is overlaid on the top of normal system memory, the CPU output signal SMIACT# must be used to distinguish SMRAM from main system memory. Additionally, both the CPU internal cache and any second level caches must be empty before the first read of an SMM handler routine and before the first read of normal memory following an SMM handler routine. This is done by flushing the caches, and is required to maintain cache coherency. When the default SMRAM location

is used, SMRAM is overlaid on top of system main memory (at 38000H through 3FFFFH).

If SMRAM is located in its own distinct memory space, which can be completely decoded with only the CPU address signals, it is said to be non-overlaid. In this case, there are no new requirements for maintaining cache coherency.

4.6.2 CACHE FLUSHES

The CPU does not unconditionally flush its cache before entering SMM (this option is left to the system designer). If SMRAM is shadowed in a memory area that is visible to the application or operating system, it is necessary for the system to empty both the CPU cache and any second level cache before entering and after leaving SMM. That is, if SMRAM is in the same physical address location as the normal cacheable memory space, then an SMM read may HIT the cache which would contain normal memory space code/data. Likewise the normal read cycles after SMM may HIT the cache which may contain SMM code/data. In this case the cache should be empty before the first memory read cycle during SMM and before the first normal cycle after exiting SMM.

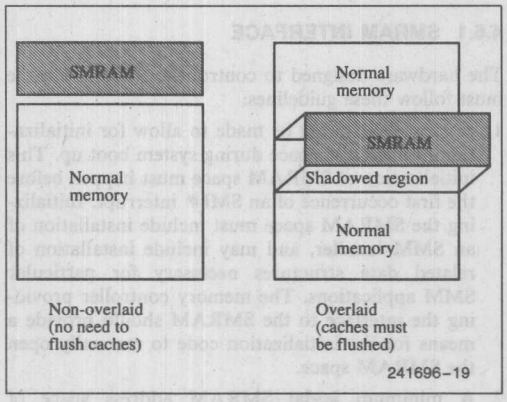


Figure 4-8. SMRAM Location

The FLUSH# and KEN# signals can be used to ensure cache coherency when switching between normal and SMM modes. Cache flushing during SMM entry is accomplished by asserting the FLUSH# pin when SMIACT# is driven active. Cache flushing during SMM exit is accomplished by asserting the FLUSH# pin after the SMIACT# pin is de-asserted (within 1 CLK). To guarantee this behavior, the constraints on setup and hold timings on the interaction of FLUSH# and SMIACT# as specified for a processor should be followed.

If the SMRAM area is overlaid over normal memory and if the system designer does not want to flush the caches upon leaving SMM then references to the SMRAM area should not be cached. It is the obligation of the system designer to ensure that the KEN# pin is sampled inactive during all references to the SMRAM area.

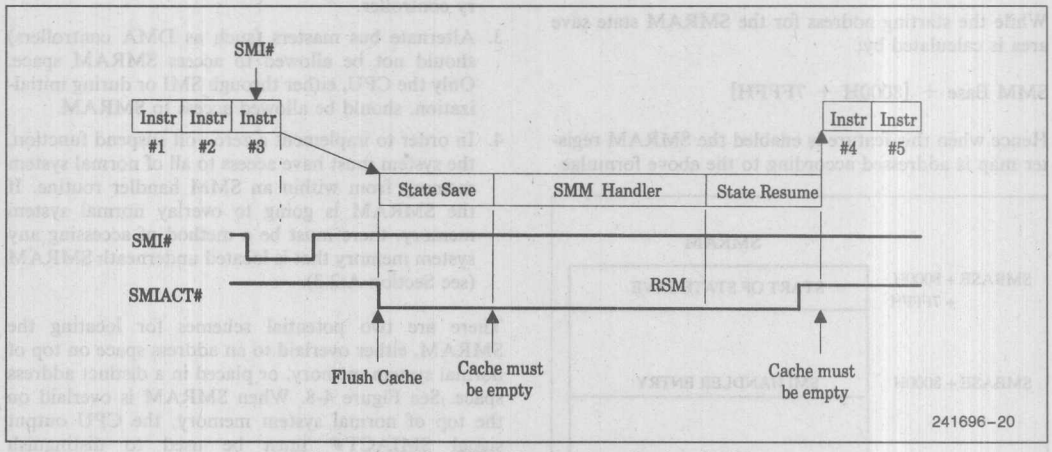


Figure 4-9. FLUSH# Mechanism During SMM

Two methods are suggested as shown in Figures 4-10 and 4-11.

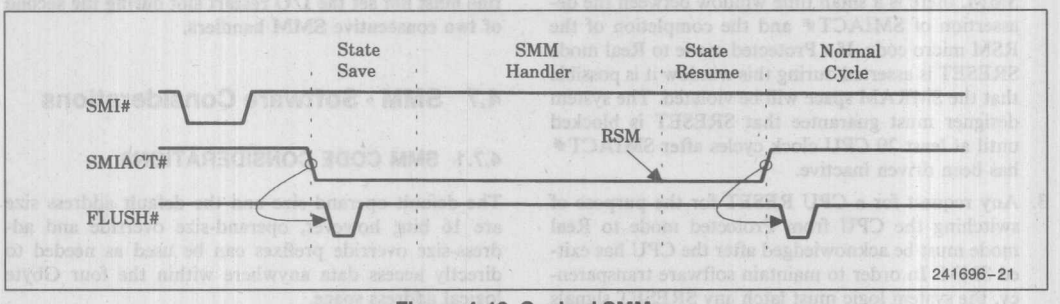


Figure 4-10. Cached SMM

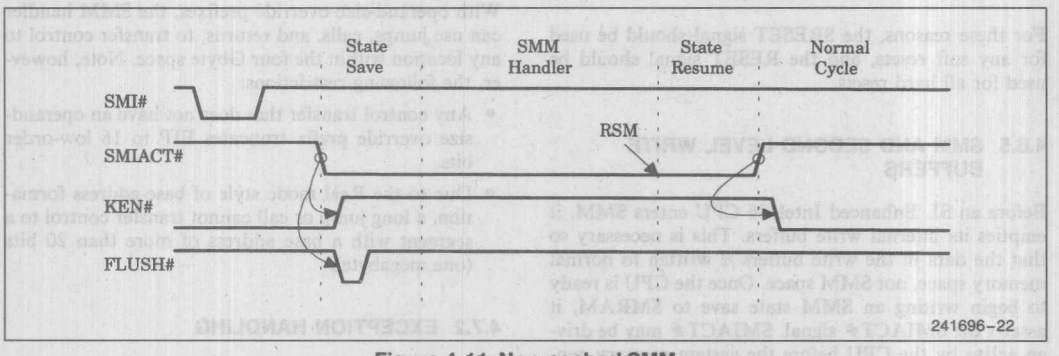


Figure 4-11. Non-cached SMM

4.6.3 A20M# PIN AND SMBASE RELOCATION

Systems based on the MS-DOS operating system contain a feature that enables the CPU address bit A20 to be forced to 0. This limits physical memory to a maximum of 1 Mbyte, and is provided to ensure compatibility with those programs that relied on the physical address wrap around functionality of the original IBM PC. The A20M# pin on SL Enhanced Intel486 CPUs provides this function. When A20M# is active, all external bus cycles will drive A20 low, and all internal cache accesses will be performed with A20 low.

The A20M# pin is recognized while the CPU is in SMM. The functionality of the A20M# input must be recognized in two instances:

1. If the SMM handler needs to access system memory space above 1 Mbyte (for example, when saving memory to disk for a zero-volt suspend), the A20M# pin must be deasserted before the memory above 1 Mbyte is addressed.
2. If SMRAM has been relocated to address space above 1 Mbyte, and A20M# is active upon entering SMM, the CPU will attempt to access SMRAM at the relocated address, but with A20 low. This could cause the system to crash, since there would be no valid SMM interrupt handler at the accessed location.

In order to account for these two situations, the system designer must ensure that A20M# is deasserted on entry to SMM. A20M# must be driven inactive before the first cycle of the SMM state save, and must be returned to its original level after the last cycle of the SMM state restore. This can be done by blocking the assertion of A20M# whenever SMIACT# is active.

4.6.4 CPU RESET DURING SMM

The system designer should take into account the following restrictions while implementing the CPU Reset logic.

1. When running software written for the 80286 CPU, a CPU RESET is used to switch the CPU from Protected mode to Real mode. If we look at the relative interrupt priorities, RESET has higher priority than SMI#. When the CPU is in SMM, the SRESET to the CPU during SMM should be blocked until the CPU exits SMM. SRESET must be blocked beginning from the time when SMI# is driven active. Care should be taken not to block the global system RESET, which may be necessary to recover from a system crash.

2. During execution of the RSM instruction to exit SMM, there is a small time window between the deassertion of SMIACT# and the completion of the RSM micro code. If a Protected mode to Real mode SRESET is asserted during this window it is possible that the SMRAM space will be violated. The system designer must guarantee that SRESET is blocked until at least 20 CPU clock cycles after SMIACT# has been driven inactive.
3. Any request for a CPU RESET for the purpose of switching the CPU from Protected mode to Real mode must be acknowledged after the CPU has exited SMM. In order to maintain software transparency, the system logic must latch any SRESET signals which are blocked during SMM.

For these reasons, the SRESET signal should be used for any soft resets, and the RESET signal should be used for all hard resets.

4.6.5 SMM AND SECOND LEVEL WRITE BUFFERS

Before an SL Enhanced Intel486 CPU enters SMM, it empties its internal write buffers. This is necessary so that the data in the write buffers is written to normal memory space, not SMM space. Once the CPU is ready to begin writing an SMM state save to SMRAM, it asserts the SMIACT# signal. SMIACT# may be driven active by the CPU before the system memory controller has had an opportunity to empty the second level write buffers.

To prevent the data from these second level write buffers from being written to the wrong location, the system memory controller needs to direct the memory write cycles to either SMM space or normal memory space. This can be accomplished by saving the status of SMIACT# along with the address for each word in the write buffers.

4.6.6 NESTED SMI# AND I/O RESTART

Special care must be taken when executing an SMM handler for the purpose of restarting an I/O instruction. When the CPU executes a RSM instruction with the I/O restart slot set, the restored EIP is modified to point to the instruction immediately preceding the SMI# request, so that the I/O instruction can be re-executed. If a new SMI# request is received while the CPU is executing an SMM handler, the CPU will service this SMI# request before restarting the original I/O instruction. If the I/O restart slot is set when the CPU executes the RSM instruction for the second SMM handler, the RSM micro code will decrement the restored EIP again. EIP now points to an address different from the originally interrupted instruction, and the CPU will begin execution of the interrupted application code at an incorrect entry point.

To prevent this from occurring, the SMM handler routine must not set the I/O restart slot during the second of two consecutive SMM handlers.

4.7 SMM - Software Considerations

4.7.1 SMM CODE CONSIDERATIONS

The default operand size and the default address size are 16 bits; however, operand-size override and address-size override prefixes can be used as needed to directly access data anywhere within the four Gbyte logical address space.

With operand-size override prefixes, the SMM handler can use jumps, calls, and returns, to transfer control to any location within the four Gbyte space. Note, however, the following restrictions:

- Any control transfer that does not have an operand-size override prefix truncates EIP to 16 low-order bits.
- Due to the Real mode style of base-address formation, a long jump or call cannot transfer control to a segment with a base address of more than 20 bits (one megabyte).

4.7.2 EXCEPTION HANDLING

Upon entry into SMM, external interrupts that require handlers are disabled (the IF bit in the EFLAGS is cleared). This is necessary because, while the processor is in SMM, it is running in a separate memory space. Consequently the vectors stored in the interrupt descriptor table (IDT) for the prior mode are not applicable. Before allowing exception handling (or software interrupts), the SMM program must initialize new interrupt and exception vectors. The interrupt vector table for SMM has the same format as for Real mode. Until the interrupt vector table is correctly initialized, the SMM handler must not generate an exception (or software interrupt). Even though hardware interrupts are disabled, exceptions and software interrupts can still occur. Only a correctly written SMM handler can prevent internal exceptions. When new exception vectors are initialized, internal exceptions can be serviced. The following are the restrictions:

1. Due to the Real mode style of base address formation, an interrupt or exception cannot transfer control to a segment with a base address of more than 20 bits.
2. An interrupt or exception cannot transfer control to a segment offset of more than 16 bits (64 KBytes).
3. If exceptions or interrupts are allowed to occur, only the low order 16 bits of the return address (EIP) are pushed onto the stack. If the offset of the interrupted procedure is greater than 64 KBytes, it is not

possible for the interrupt/exception handler to return control to that procedure. (One work-around could be to perform software adjustment of the return address on the stack).

4. The SMBASE Relocation feature affects the way the CPU will return from an interrupt or exception during an SMI handler.

4.7.3 HALT DURING SMM

HALT should not be executed during SMM, unless interrupts have been enabled (see section 4.7.2). Interrupts are disabled in SMM and INTR, NMI, and SMI# are the only events that take the CPU out of HALT.

4.7.4 RELOCATING SMRAM TO AN ADDRESS ABOVE ONE MEGABYTE

Within SMM (or Real mode), the segment base registers can only be updated by changing the segment register. The segment registers contain only 16 bits, which allows only 20 bits to be used for a segment base address (the segment register is shifted left four bits to determine the segment base address). If SMRAM is relocated to an address above one megabyte, the segment registers can no longer be initialized to point to SMRAM.

These areas can still be accessed by using address override prefixes to generate an offset to the correct address. For example, if the SMBASE has been relocated immediately below 16M, the DS and ES registers are still initialized to 0000 0000H. We can still access data in SMRAM by using 32-bit displacement registers:

```
mov esi,00FFxxxxH ;64K segment immediately
                  ;below 16M
mov ax,ds:[esi]
```

5.0 RESET AND INITIALIZATION

5.1 RESET

The RESET input must be used at power-up to initialize the CPU. The Reset input forces the CPU to begin execution at a known state. The microprocessor cannot begin execution of instructions until at least 1 ms after V_{CC} and CLK have reached their proper D.C. and A.C. specifications. The RESET pin should remain active during this time to ensure proper microprocessor operation. However, for warm boot-ups RESET should remain active for at least 15 CLK periods. RESET is active HIGH. RESET is asynchronous but must meet setup and hold times t20 and t21 for recognition in any specific clock. The RESET signal is the same as the standard Intel486 CPU RESET signal with the following exceptions.

RESET has a special function of resetting SMBASE to the default value of 30000H. If SMBASE relocation is not used, the RESET signal can be used as the only reset.

The microprocessor will be placed in the Power Down mode if UP# is sampled active at the falling edge of RESET.

5.2 SRESET

The SRESET (Soft RESET) input, has the same functions as RESET, but does not change the SMBASE, and UP# is not sampled on the falling edge of SRESET. If SMBASE relocation is used by the system, the soft resets should be handled using the SRESET input. The SRESET signal should not be used for the cold boot-up power-on reset.

The SRESET input pin is provided to save the status of SMBASE during i286 CPU-compatible mode change. SRESET leaves the status of the on-chip FPU and SMBASE intact while resetting other units including the on-chip cache. For compatibility with future generation Intel486 CPUs, the system should not rely on flushing the on-chip cache through the SRESET input pin. The FLUSH# input pin is provided for this purpose and should be used to flush the on-chip cache.

SRESET should not be used to initiate test modes (this behavior is subject to change in future versions).

5.3 Pin State During Reset

While in reset, the SL Enhanced Intel486 microprocessor bus is in the state shown in Figure 5-1, if the HOLD, AHOLD and BOFF# requests are inactive. Note that the address (A31-A2, BE3#-BE0#) and cycle definition (M/IO#, D/C#, W/R#) pins are undefined from the time reset is asserted up to the start of the first bus cycle. All defined pins (except FERR#) assume known values at the beginning of the first bus cycle. The first bus cycle is always a code fetch to address 0FFFFFFF0H. FERR# reflects the state of the ES (error summary status) bit in the floating point unit status word. The ES bit is initialized whenever the floating point unit state is initialized.

Figures 5-1 and 5-2 show the bus state for the SL Enhanced Intel486 DX and SX CPUs when UP# is not asserted and for the SL Enhanced Intel486 SX CPU when UP# is asserted.

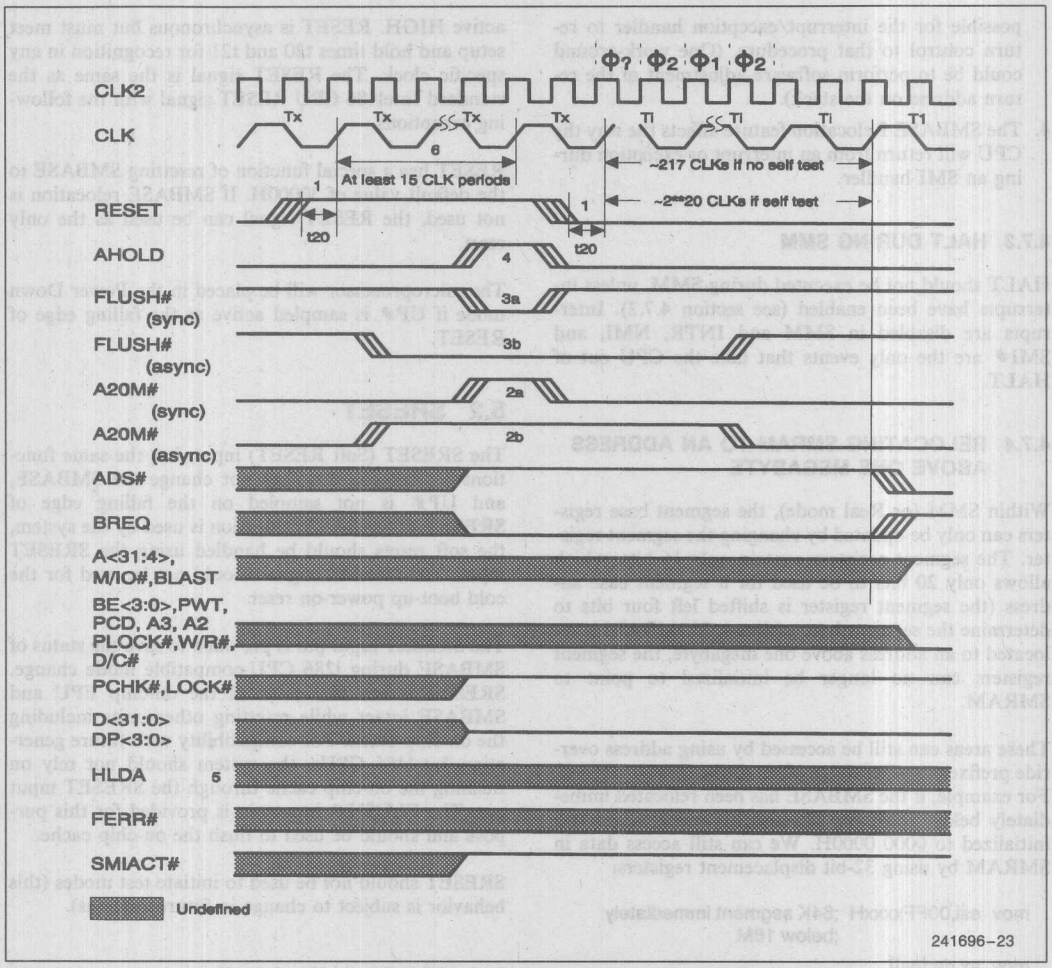
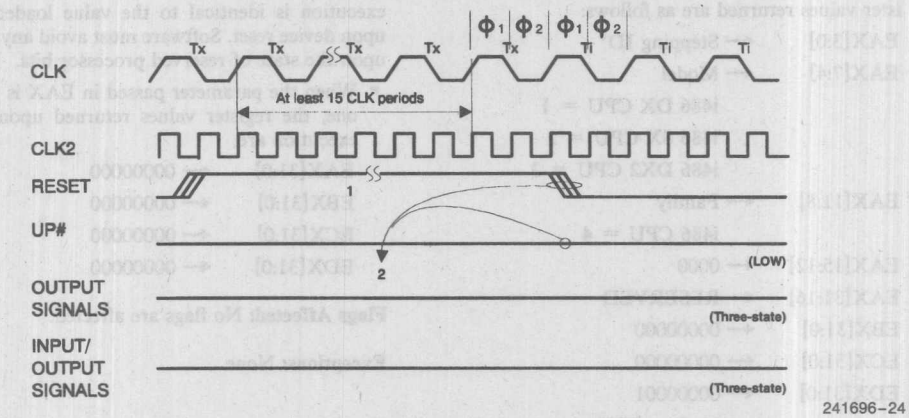


Figure 5-1. SL Enhanced Intel486™ DX CPU or SL Enhanced Intel486 SX CPU without UP# Asserted During SRESET or RESET

NOTES:
1. RESET is an asynchronous input. t_{20} must be met only to guarantee recognition on a specific clock edge.
2a. When A20M# is driven synchronously, it must be driven high (inactive) for the CLK edge prior to and the falling edge of RESET to ensure proper operation. A20M# setup and hold times must be met.
2b. When A20M# is driven asynchronously, it must be driven high (inactive) for two CLKs prior to and two CLKs after the falling edge of RESET to ensure proper operation.
3a. When FLUSH# is driven synchronously, it should be driven low (active) for the CLK edge prior to the falling edge of RESET to invoke the Three-State Output Test Mode. All outputs are guaranteed three-stated within 10 CLKs of RESET being deasserted. FLUSH# setup and hold times must be met.
3b. When FLUSH# is driven asynchronously, it must be driven low (active) for two CLKs prior to and two CLKs after the falling edge of RESET to invoke the Three-State Output Test Mode. All outputs are guaranteed three-stated within 10 CLKs of RESET being deasserted.
4. AHOLD should be driven high (active) for the CLK edge prior to the falling edge of RESET to invoke the Build-In-Self-Test (BIST). AHOLD setup and hold times must be met.
5. Hold is recognized normally during RESET.
6. 15 CLKs RESET pulse width for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.



1. 15 CLKs reset pulse width for warm reset. Power-up resets require reset to be asserted for at least 1 ms after V_{CC} and CLK are stable.
2. When UP# is active, the falling edge of RESET will disable all the internal input buffers except UP#. RESET and the internal input buffers of CLK will also be disabled.

Figure 5-2. SL Enhanced Intel486™ SX CPU During SRESET or RESET with UP# Asserted

5.4 CPU Identification Codes

The DX register always contains a component identifier at the conclusion of RESET. The upper byte of DX (DH) will contain 04 and the lower byte of DX (DL) will contain a CPU type/stepping identifier.

Table 5-1. CPU ID Codes

SL Enhanced Intel486™ CPU	Component ID (DH)	Revision ID (DL)
Intel486 SX CPU	04	2x
Intel486 DX CPU	04	1x
Intel486 DX2 CPU	04	3x

5.4.1 CPU ID INSTRUCTION

The SL Enhanced Intel486 microprocessor family implements a new instruction that makes information available to software about the family, model and stepping of the microprocessor on which it is executing. Support of this instruction is indicated by the presence of a user-modifiable bit in position EFLAGS.21, referred to as the EFLAGS.ID bit. The actual state of the EFLAGS.ID bit is irrelevant and provides no significance to the hardware. This bit is reset to zero upon device reset (RESET or SRESET) for compatibility with existing Intel486 processor designs.

Table 5-2. CPUID Instruction Description

OP CODE	Instruction	CPU Core Clocks	EAX Input Value	Description
0F A2	CPUID	14	1	CPU Identification
		9	0 or >1	Intel string/ null registers

Operation:

The CPUID instruction requires the user to pass an input parameter to the CPU in the EAX register. The CPU response is returned to the user in registers EAX, EBX, ECX and EDX.

- When the parameter passed in EAX is zero, the register values returned upon instruction execution are:

  EAX[31:0] ← 1
  EBX[31:0] ← 756E6547
  ECX[31:0] ← 6C65746E
  EDX[31:0] ← 49656E69

The values in EBX, ECX, and EDX indicate an Intel microprocessor. When taken in the proper order, they decode to the string "GenuineIntel".

- When the parameter passed in EAX is one, the register values returned are as follows:

EAX[3:0]	← Stepping ID*
EAX[7:4]	← Model
	i486 DX CPU = 1
	i486 SX CPU = 2
	i486 DX2 CPU = 3
EAX[11:8]	← Family
	i486 CPU = 4
EAX[15:12]	← 0000
EAX[31:16]	← RESERVED
EBX[31:0]	← 00000000
ECX[31:0]	← 00000000
EDX[31:0]	← 00000001
	bit 0 = FPU

*Please contact Intel for stepping ID details.

The value returned in EAX after CPUID instruction execution is identical to the value loaded into EDX upon device reset. Software must avoid any dependency upon the state of reserved processor bits.

- When the parameter passed in EAX is greater than one, the register values returned upon instruction execution are:

EAX[31:0]	← 00000000
EBX[31:0]	← 00000000
ECX[31:0]	← 00000000
EDX[31:0]	← 00000000

Flags Affected: No flags are affected.

Exceptions: None.

6.0 ELECTRICAL AND MECHANICAL SPECIFICATIONS

6.1 D.C. Specifications

6.1.1 3.3V D.C. CHARACTERISTICS

Table 6-1. 3.3V D.C. Specifications

Functional operating range: V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C

Symbol	Parameter	Min	Typ	Max	Unit	Test Cond.
V_{IL}	Input LOW Voltage	−0.3		+0.8	V	
V_{IH}	Input HIGH Voltage	2.0		V_{CC}+0.3	V	Note 4
V_{IHC}	Input HIGH Voltage of CLK, CLK2	V_{CC}−0.6		V_{CC}+0.3	V	
V_{OL}	Output LOW Voltage I_{OL} = 2.0 mA I_{OL} = 100 uA			 0.4 0.2	 V V	
V_{OH}	Output HIGH Voltage I_{OH} = −2.0 mA I_{OH} = −100 uA	 2.4 V_{CC}−0.2			 V V	
I_{CCU}	UP# Active Supply Current		15	35	mA	Note 5
I_{LI}	Input Leakage Current	−15		15	µA	Note 1
I_{IH}	Input Leakage Current			200	µA	Note 2
I_{IL}	Input Leakage Current			−400	µA	Note 3
I_{LO}	Output Leakage Current	−15		15	µA	
C_{IN}	Input Capacitance			10	pF	Note 6
C_{OUT}	Output or I/O Capacitance			10	pF	Note 6
C_{CLK}	CLK Capacitance			6	pF	Note 6

NOTES:
1. This parameter is for inputs without pull-ups or pull downs and $0V < V_{IN} < V_{CC}$.
2. This parameter is for inputs with Pull downs and V_{IH} = 2.4V.
3. This parameter is for inputs with pull-ups and V_{IL} = 0.4V.
4. All inputs except CLK, CLK2.
5. When the CPU is in Stop Grant state, the I_{CCU} of the host CPU is less than 2 mA.
6. F_C=1 MHz; Not 100% tested.

Table 6-2. 3.3V Preliminary I_{CC} Values for 1X Clock SL Enhanced Intel486 SX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
3.3V	I_{CC} Active	25 MHz	250 mA	315 mA
		33 MHz	300 mA	385 mA
	I_{CC} Stop Grant	25 MHz	20 mA	40 mA
		33 MHz	25 mA	50 mA
	I_{CC} Stop Clock(1)	0 MHz	100 μA	1 mA

Table 6-3. 3.3V Preliminary I_{CC} Values for 1X Clock SL Enhanced Intel486 DX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
3.3V	I_{CC} Active	33 MHz	330 mA	415 mA
	I_{CC} Stop Grant	33 MHz	25 mA	50 mA
	I_{CC} Stop Clock(1)	0 MHz	100 μA	1 mA

Table 6-4. 3.3V Preliminary I_{CC} Values for SL Enhanced Intel486 DX2 CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
3.3V	I_{CC} Active	40 MHz	375 mA	450 mA
		50 MHz	460 mA	550 mA
	I_{CC} Stop Grant	40 MHz	20 mA	40 mA
		50 MHz	23 mA	50 mA
	I_{CC} Stop Clock(1)	0 MHz	100 μA	1 mA

NOTE:
1. The V_{IH} and V_{IL} levels must be equal to V_{CC} and 0V, respectively, in order to meet the I_{CC} Stop Clock specifications.

6.1.2 5V D.C. CHARACTERISTICS

Table 6-5. 5V D.C. Specifications

Functional operating range: V_{CC} = 5V ± 5%; T_{CASE} = 0°C to +85°C

Symbol	Parameter	Min	Typical	Max	Unit	Test Condition
V_{IL}	Input LOW Voltage	−0.3		+0.8	V	
V_{IH}	Input HIGH Voltage	2.0		V_{CC}+0.3	V	
V_{OL}	Output LOW Voltage			0.45	V	Note 1
V_{OH}	Output HIGH Voltage	2.4			V	Note 2
I_{CCU}	UP# Active Supply Current		25	50	mA	Note 6
I_{LI}	Input Leakage Current	−15		15	µA	Note 3
I_{IH}	Input Leakage Current			200	µA	Note 4
I_{IL}	Input Leakage Current			−400	µA	Note 5
I_{LO}	Output Leakage Current	−15		15	µA	
C_{IN}	Input Capacitance PGA PQFP			 20 10	 pF pF	Note 7
C_{OUT}	Output or I/O Capacitance PGA PQFP			 20 10	 pF pF	Note 7
C_{CLK}	CLK Capacitance PGA PQFP			 20 6	 pF pF	Note 7

NOTES:
1. This parameter is measured at: Address, Data, BEn 4.0 mA; Definition, Control 5.0 mA
2. This parameter is measured at: Address, Data, BEn −1.0 mA; Definition, Control −0.9 mA
3. This parameter is for inputs without pull-ups or pull-downs and $0V \le V_{IN} \le V_{CC}$.
4. This parameter is for inputs with pull-downs and V_{IH} = 2.4V.
5. This parameter is for inputs with pull-ups and V_{IL} = 0.45V.
6. When the CPU is in Stop Grant state, the I_{CCU} of the host CPU is less than 2 mA.
7. F_C=1 MHz; Not 100% tested.

Table 6-6. 5V Preliminary I_{CC} Values for 1X Clock SL Enhanced Intel486 SX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
5V	I_{CC} Active	25 MHz	430 mA	560 mA
		33 MHz	590 mA	685 mA
	I_{CC} Stop Grant	25 MHz	35 mA	65 mA
		33 MHz	40 mA	80 mA
	I_{CC} Stop Clock(1)	0 MHz	200 µA	2 mA

Table 6-7. 5V Preliminary I_{CC} Values for 1X Clock SL Enhanced Intel486 DX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
5V	I_{CC} Active	33 MHz	500 mA	630 mA
		50 MHz	775 mA	950 mA
	I_{CC} Stop Grant	33 MHz	40 mA	80 mA
		50 MHz	55 mA	100 mA
	I_{CC} Stop Clock(1)	0 mHz	200 µA	2 mA

Table 6-8. 5V Preliminary I_{CC} Values for SL Enhanced Intel486 DX2 CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
5V	I_{CC} Active	50 MHz	775 mA	950 mA
		66 MHz	975 mA	1200 mA
	I_{CC} Stop Grant	50 MHz	35 mA	70 mA
		66 MHz	45 mA	90 mA
	I_{CC} Stop Clock(1)	0 MHz	200 µA	2 mA

NOTE:
1. The V_{IH} and V_{IL} levels must be equal to V_{CC} and 0V, respectively, in order to meet the I_{CC} Stop Clock specifications.

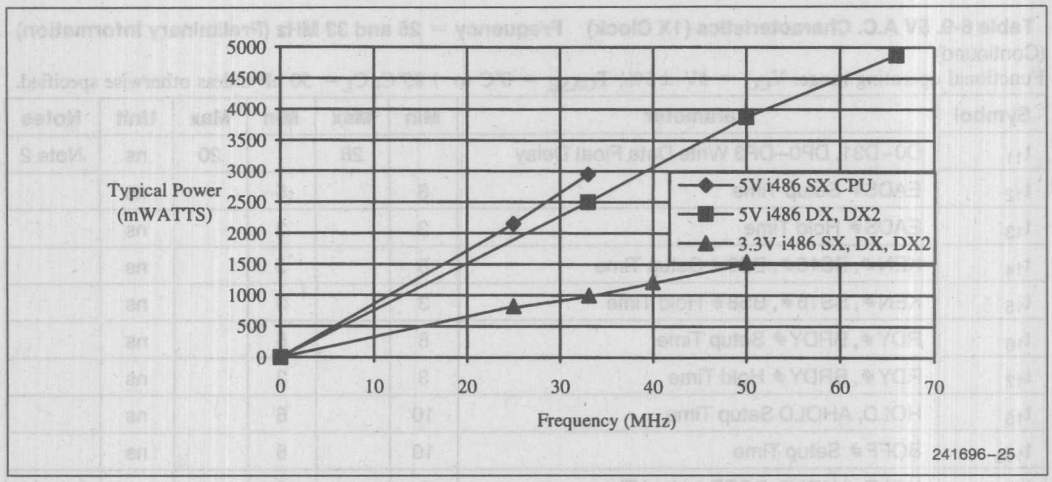


Figure 6-1. Frequency vs Power (Typ) in 1X CLK Mode for SL Enhanced Intel486 SX, DX and DX2 CPUs

6.2 A.C. Specifications for 1X CLK Option

6.2.1 5V A.C. CHARACTERISTICS

The A.C. specifications given in the tables of this section consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the rising edge of the input system clock (CLK) unless otherwise specified.

Table 6-9. 5V A.C. Characteristics (1X Clock) Frequency = 25 and 33 MHz (Preliminary Information)
Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Min	Max	Unit	Notes
	Frequency	8	**25**	8	**33**	MHz	Note 1
t_1	CLK Period	40	125	30	125	ns	
t_{1a}	CLK Period Stability		0.1		0.1	%	Adjacent clocks
t_2	CLK High Time	14		11		ns	at 2V
t_3	CLK Low Time	14		11		ns	at 0.8V
t_4	CLK Fall Time		4		3	ns	2V to 0.8V
t_5	CLK Rise Time		4		3	ns	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA, SMIACT#, **FERR#** * Valid Delay	3	19	3	16	ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		28		20	ns	Note 2
t_8	PCHK# Valid Delay	3	24	3	22	ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	24	3	20	ns	
t_9	BLAST#, PLOCK# Float Delay		28		20	ns	Note 2
t_{10}	D0–D31, DP0–DP3 Write Data Valid Delay	3	20	3	18	ns	

Table 6-9. 5V A.C. Characteristics (1X Clock) Frequency = 25 and 33 MHz (Preliminary Information) (Continued)

Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Min	Max	Unit	Notes
t_{11}	D0–D31, DP0–DP3 Write Data Float Delay		28		20	ns	Note 2
t_{12}	EADS# Setup Time	8		5		ns	
t_{13}	EADS# Hold Time	3		3		ns	
t_{14}	KEN#, BS16#, BS8# Setup Time	8		5		ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		3		ns	
t_{16}	RDY#, BRDY# Setup Time	8		5		ns	
t_{17}	RDY#, BRDY# Hold Time	3		3		ns	
t_{18}	HOLD, AHOLD Setup Time	10		6		ns	
t_{18a}	BOFF# Setup Time	10		8		ns	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		3		ns	
t_{20}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#*** Setup Time	10		5		ns	
t_{21}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#*** Hold ime	3		3		ns	
t_{22}	D0–D31, DP0–DP3, A4–A31 Read Setup Time	5		5		ns	
t_{23}	D0–D31, DP0–DP3, A4–A31 Read Hold Time	3		3		ns	

NOTES:
*Present only in the SL Enhanced Intel486 DX and DX2 CPUs.
1. 0 MHz operation is guaranteed when the STPCLK# and STOP GRANT bus cycle protocol is used.
2. Not 100% tested, guaranteed by design characterization.

Table 6-10. 5V A.C. Characteristics (1X Clock) Frequency = 50 MHz (Intel486 DX CPU) (Preliminary Information)

Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = 0 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Unit	Notes
	Frequency	16	50	MHz	Note 1
t_1	CLK Period	20	62.5	ns	
t_{1a}	CLK Period Stability		0.1	%	Adjacent clocks
t_2	CLK High Time	7		ns	at 2V
t_3	CLK Low Time	7		ns	at 0.8V
t_4	CLK Fall Time		2	ns	2V to 0.8V
t_5	CLK Rise Time		2	ns	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA, SMIACT#, FERR# Valid Delay	3	12	ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		18	ns	Note 2
t_8	PCHK# Valid Delay	3	14	ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	12	ns	
t_9	BLAST#, PLOCK# Float Delay		18	ns	Note 2
t_{10}	D0–D31, DP0–DP3 Write Data Valid Delay	3	12	ns	
t_{11}	D0–D31, DP0–DP3 Write Data Float Delay		18	ns	Note 2
t_{12}	EADS# Setup Time	5		ns	
t_{13}	EADS# Hold Time	2		ns	
t_{14}	KEN#, BS16#, BS8# Setup Time	5		ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	2		ns	
t_{16}	RDY#, BRDY# Setup Time	5		ns	
t_{17}	RDY#, BRDY# Hold Time	2		ns	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	5		ns	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	2		ns	
t_{20}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, IGNNE# Setup Time	5		ns	
t_{21}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, IGNNE# Hold Time	2		ns	
t_{22}	D0–D31, DP0–DP3, A4–A31 Read Setup Time	4		ns	
t_{23}	D0–D31, DP0–DP3, A4–A31 Read Hold Time	2		ns	

NOTES:
1. 0 MHz operation is guaranteed when the STPCLK# and STOP GRANT bus cycle protocol is used.
2. Not 100% tested, guaranteed by design characterization.

The following specifications are different for existing Intel486 DX2 CPU products. A system board that will support all of the SL Enhanced Intel486 CPU products should be designed to the worst-case specifications of the 25 and 33 MHz local bus timings.

Table 6-11. A.C. Characteristics (1X Clock) Frequency = 50 and 66 MHz (Intel486 DX2 CPU) (Preliminary Information)

Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

		DX2-50		DX2-66		
Symbol	**Parameter**	**Min**	**Max**	**Min**	**Max**	**Unit**
	Frequency		**50**		**66**	MHz
	CLK Frequency	8	**25**	8	**33**	MHz
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA, SMIACT#, **FERR#** Valid Delay				14	ns
t_8	PCHK# Valid Delay				14	ns
t_{8a}	BLAST#, PLOCK# Valid Delay				14	ns
t_{10}	D0–D31, DP0–DP3 Write Data Valid Delay				14	ns
t_{18}	HOLD, AHOLD Setup Time	8				ns
t_{18a}	BOFF# Setup Time	8		7		ns
t_{20}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#** Setup Time	8				ns

NOTES:
1. 0 MHz operation is guaranteed when the STPCLK# and STOP GRANT bus cycle protocol is used.
2. Not 100% tested, guaranteed by design characterization.

6.2.2 3.3V A.C. CHARACTERISTICS

**Table 6-12. 3.3V A.C. Characteristics (1X Clock Option)
Frequency = 20 (Intel486 DX2-40 CPU), 25, (Intel486
DX2-50 and SX-25 CPUs) and 33 MHz. (Preliminary Information)**

Functional operating range: V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Min	Max	Min	Max	Figure	Unit	Notes
	Frequency	8	**20**	8	**25**	8	**33**		MHz	Note 1
t_1	CLK Period	50	125	40	125	30	125		ns	
t_{1a}	CLK Period Stability		0.1		0.1		0.1		%	Adjacent clocks
t_2	CLK High Time	16		14		11			ns	at 2V
t_3	CLK Low Time	16		14		11			ns	at 0.8V
t_4	CLK Fall Time		6		4		3		ns	2V to 0.8V
t_5	CLK Rise Time		6		4		3		ns	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA, SMIACT#, **FERR#** * Valid Delay	3	23	3	19	3	16		ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		37		28		20		ns	Note 2
t_8	PCHK# Valid Delay	3	28	3	24	3	22		ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	28	3	24	3	20		ns	
t_9	BLAST#, PLOCK# Float Delay		37		28		20		ns	Note 2
t_{10}	D0–D31, DP0–DP3 Write Data Valid Delay	3	26	3	20	3	19		ns	
t_{11}	D0–D31, DP0–DP3 Write DAta Float Delay		37		28		20		ns	Note 2
t_{12}	EADS# Setup Time	10		8		6			ns	
t_{13}	EADS# Hold Time	3		3		3			ns	
t_{14}	KEN#, BS16#, BS8# Setup Time	10		8		6			ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		3		3			ns	
t_{16}	RDY#, BRDY# Setup Time	10		8		6			ns	
t_{17}	RDY#, BRDY# Hold Time	3		3		3			ns	
t_{18}	HOLD, AHOLD Setup Time	12		10		6			ns	
t_{18a}	BOFF# Setup Time	12		10		9			ns	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		3		3			ns	

Table 6-12. 3.3V A.C. Characteristics (1X Clock Option)
Frequency = 20, (Intel486 DX2-40 CPU), 25, (Intel486 DX2-50 and SX-25 CPUs) and 33 MHz. (Preliminary Information)

Functional operating range: V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Min	Max	Min	Max	Figure	Unit	Notes
t_{20}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#*** Setup Time	12		10		6			ns	
t_{21}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#*** Hold Time	3		3		3			ns	
t_{22}	D0–D31, DP0–DP3, A4–A31 Read Setup Time	6		6		6			ns	
t_{23}	D0–D31, DP0–DP3, A4–A31 Read Hold Time	3		3		3			ns	

NOTES:
* Present only in the SL Enhanced Intel486 DX and DX2 CPUs.
1. 0 MHz operation is guaranteed when the STPCLK# and STOP GRANT bus cycle protocol is used.
2. Not 100% tested, guaranteed by design characterization.

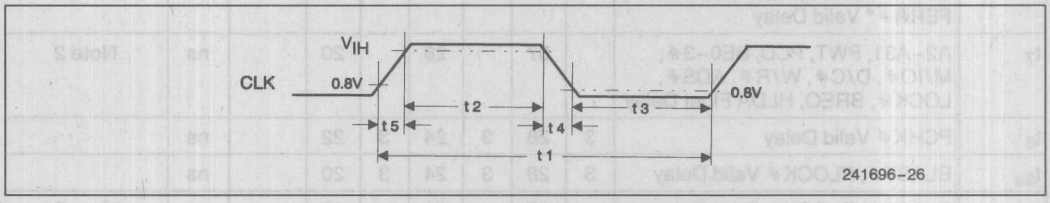


Figure 6-2. CLK Waveforms

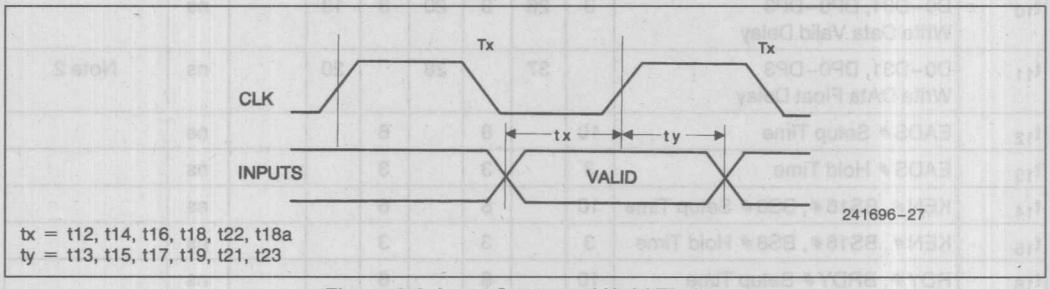


Figure 6-3. Input Setup and Hold Timing

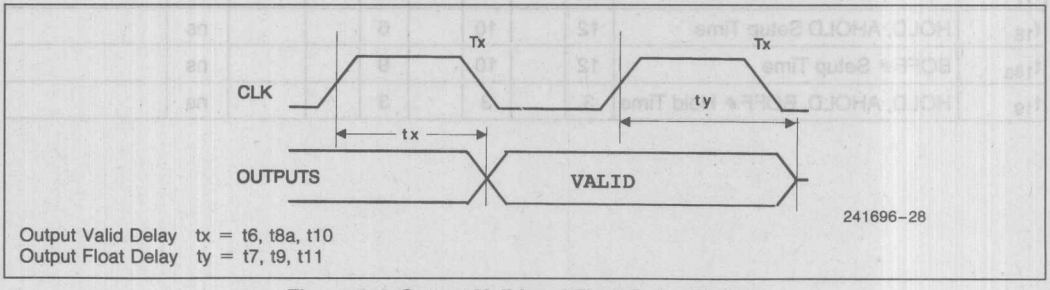


Figure 6-4. Output Valid and Float Delay Timing

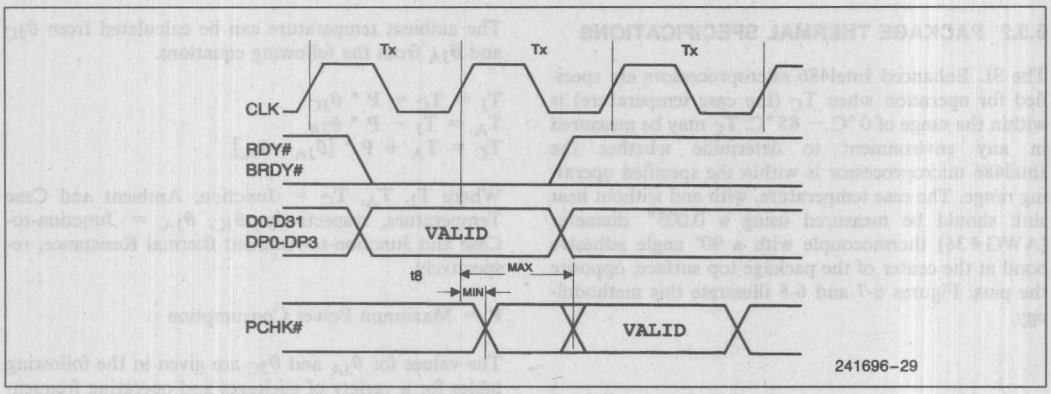


Figure 6-5. PCHK# Valid Delay Timing

6.3 Mechanical Data

6.3.1 PACKAGE MECHANICAL SPECIFICATIONS FOR THE 208 LEAD SQFP PACKAGE

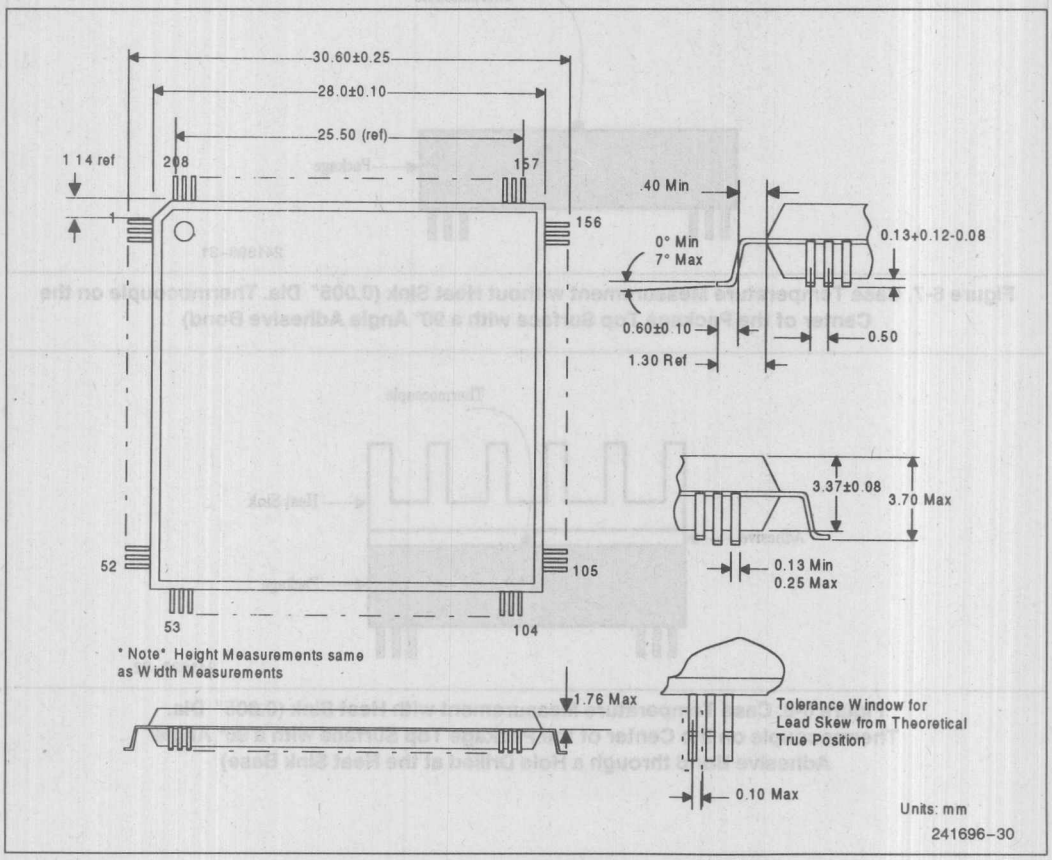


Figure 6-6. 208 Lead SQFP Package Dimensions

6.3.2 PACKAGE THERMAL SPECIFICATIONS

The SL Enhanced Intel486 microprocessors are specified for operation when T_C (the case temperature) is within the range of 0 °C – 85 °C. T_C may be measured in any environment to determine whether the Intel486 microprocessor is within the specified operating range. The case temperature, with and without heat sink should be measured using a 0.005″ diameter (AWG#36) thermocouple with a 90° angle adhesive bond at the center of the package top surface, opposite the pins. Figures 6-7 and 6-8 illustrate this methodology.

The ambient temperature can be calculated from θ_{JC} and θ_{JA} from the following equations.

$T_J = T_C + P * \theta_{JC}$
$T_A = T_J - P * \theta_{JA}$
$T_C = T_A + P * [\theta_{JA} - \theta_{JC}]$

Where T_J, T_A, T_C = Junction, Ambient and Case Temperature, respectively. θ_{JC}, θ_{JA} = Junction-to-Case and Junction-to-Ambient thermal Resistance, respectively.

P = Maximum Power Consumption

The values for θ_{JA} and θ_{JC} are given in the following tables for a variety of packages and operating frequencies.

Refer to the OverDrive Processor socket section for OverDrive processor θ_{JA} and θ_{JC} values.

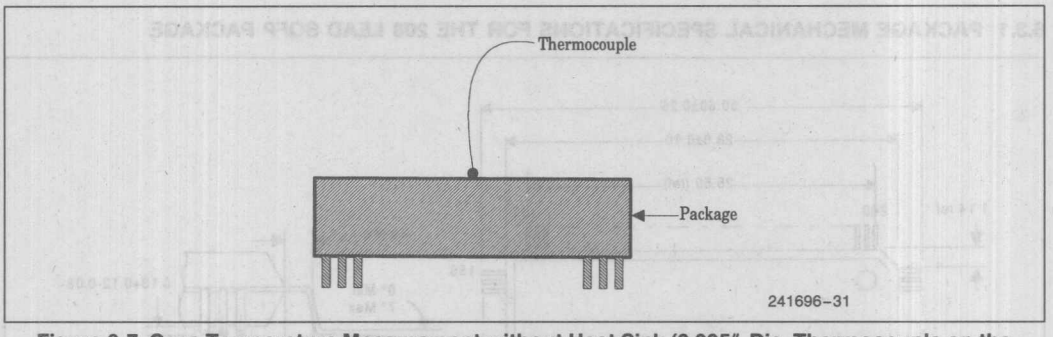


Figure 6-7. Case Temperature Measurement without Heat Sink (0.005″ Dia. Thermocouple on the Center of the Package Top Surface with a 90° Angle Adhesive Bond)

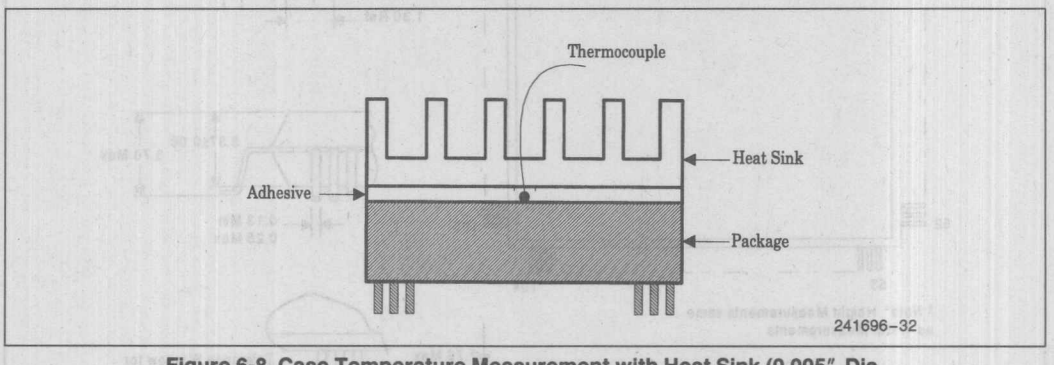


Figure 6-8. Case Temperature Measurement with Heat Sink (0.005″ Dia. Thermocouple on the Center of the Package Top Surface with a 90° Angle Adhesive Bond through a Hole Drilled at the Heat Sink Base)

Table 6-13. Thermal Resistance (°C/W) θ_{JC} and θ_{JA} for the SL Enhanced Intel486™ CPUs (for PGA)

	θ_{JC}	θ_{JA} vs. Airflow - ft/min. (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
With Heat Sink	1.5	13	8.0	6.0	5.0	4.5	4.25
Without Heat Sink	1.5	17	14.5	12.5	11.0	10.0	9.5

* 0.350″ high omnidirectional heat sink.

Table 6-14. Thermal Resistance (°C/W) θ_{JC} and θ_{JA} for the SL Enhanced Intel486™ CPUs (for PQFP)

	θ_{JC}	θ_{JA} vs. Airflow - ft/min. (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
With Heat Sink	3.5	17.0	10.5	8.5	8.0
Without Heat Sink	3.5	20.5	16.5	14.0	12.5

*0.350″ high omnidirectional heat sink.

2

Table 6-15(a). Thermal Resistance (°C/W) θ_{JA} for the SL Enhanced Intel486™ CPUs (for SQFP)

	θ_{JA} vs. Airflow - ft/min. (m/sec)			
	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
Intel486 SX CPU Without Heat Sink	36.0	27.5	25.0	22.5
Intel486 DX CPU Without Heat Sink	25.0	17.5	15.0	13.0
Intel486 DX2 CPU Without Heat Sink	24.0	17.0	15.0	13.0

Table 6-15(b). Thermal Resistance (°C/W) θ_{JC} for the SL Enhanced Intel486 CPUs (for SQFP)

	θ_{JC} vs. Airflow - ft/min. (m/sec)			
	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
Intel486 SX CPU	4.0	7.5	8.0	8.5
Intel486 DX and DX2 CPUs	3.5	6.0	6.0	6.0

The following tables show maximum ambient temperatures of SL Enhanced Intel486 CPUs for each package and operating frequency. The maximum ambient temperatures listed below are not valid for the OverDrive Processor.

Table 6-16. Maximum Tambient for the 5V, 168-pin PGA SL Enhanced Intel486™ DX and DX2 CPUs

	Freq.	Airflow - ft/min. (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
Tambient °C With Heat Sink	33	49	65	71	74	76	76
	50	30	54	64	68	71	72
	66	16	46	58	64	67	69
Tambient °C Without Heat Sink	33	36	44	50	55	58	60
	50	11	23	33	40	45	47
	66	−8	7	19	28	34	37

Table 6-17. Maximum Tambient for the 5V, 168-pin PGA SL Enhanced Intel486™ SX CPU

	Freq.	Airflow - ft/min. (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
Tambient °C With Heat Sink	25	53	67	72	75	77	77
	33	46	63	70	73	75	76
Tambient °C Without Heat Sink	25	42	49	54	58	61	63
	33	32	40	47	52	56	58

Table 6-18. Maximum Tambient for the 5V, 196-lead PQFP SL Enhanced Intel486™ DX CPU

	Freq.	Airflow - ft/min. (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
Tambient °C With Heat Sink	33	42	63	69	71
Tambient °C Without Heat Sink	33	31	44	52	57

Table 6-19. Maximum Tambient for the 5V, 196-lead PQFP SL Enhanced Intel486™ SX CPU

	Freq.	Airflow - ft/min. (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
Tambient °C With Heat Sink	25	47	65	71	72
	33	39	61	68	70
Tambient °C Without Heat Sink	25	37	49	56	60
	33	27	40	49	54

Table 6-20. Maximum Tambient for the 3.3V, 208-lead SQFP SL Enhanced Intel486™ SX CPU

	Freq.	Airflow - ft/min. (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
Tambient °C Without Heat Sink Intel486 SX CPU	25	51.1	64.0	67.0	70.0
	33	44.5	59.5	63.5	67.0

Table 6-21. Maximum Tambient for the 3.3V, 208-lead SQFP SL Enhanced Intel486™ DX CPU

	Freq.	Airflow - ft/min. (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
Tambient °C Without Heat Sink Intel486 DX CPU	33	55.5	69.0	72.5	75.5

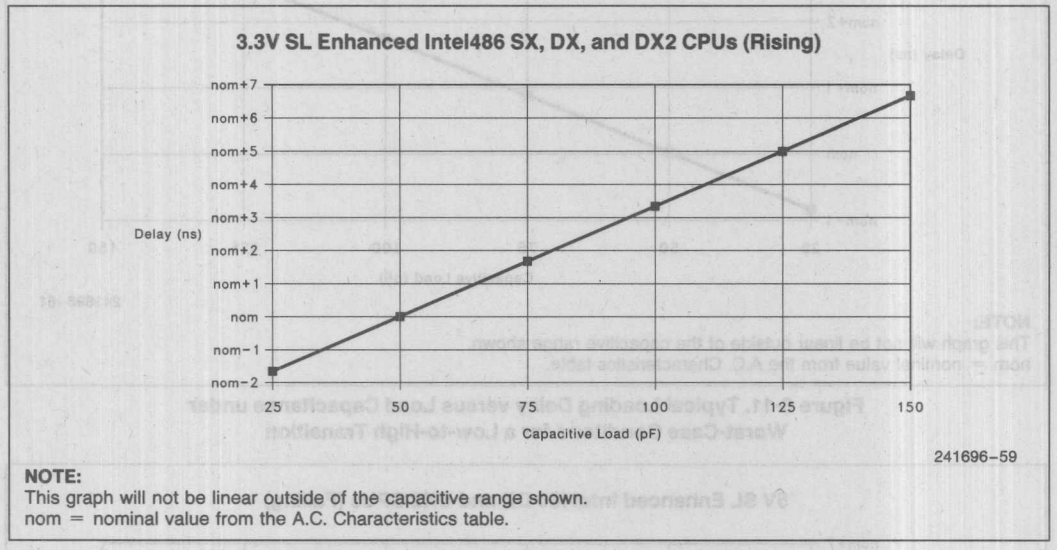
Table 6-22. Maximum Tambient for the 3.3V, 208-lead SQFP SL Enhanced Intel486™ DX2 CPU

	Freq.	Airflow - ft/min. (m/sec)			
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)
Tambient °C Without Heat Sink Intel486 DX2 CPU	40	54.5	68.5	71.5	74.5
	50	48.0	65.0	68.5	72.0

For further details about thermal and mechanical package specifications and methodologies, refer to the 1994 Packaging Handbook (order number 240800).

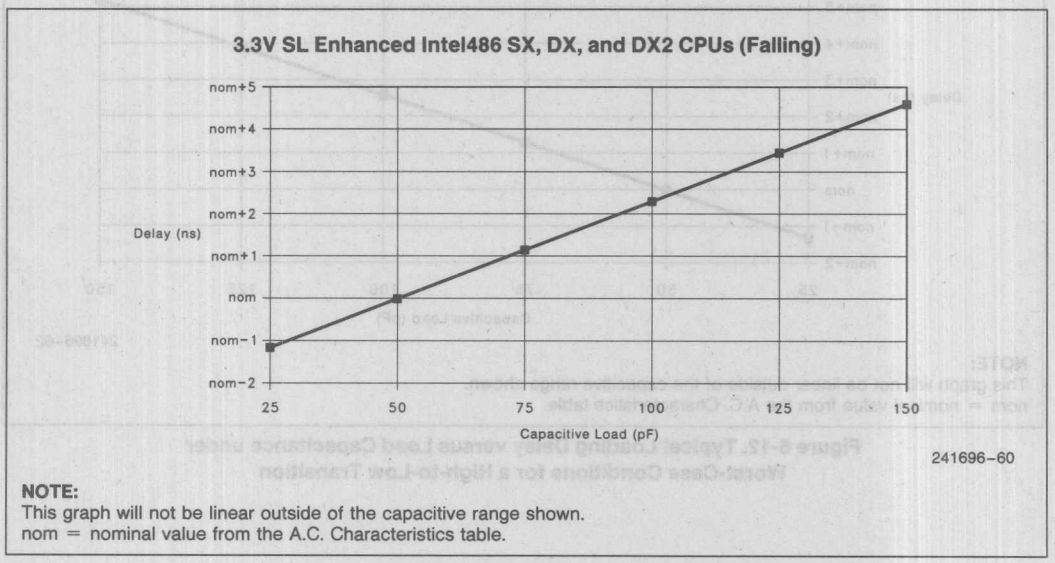
6.4 Capacitive Derating Information

The capacitive derating curves illustrate output delay versus capacitive load for 3.3V and 5V SL Enhanced Intel486 microprocessors. The derating curves show the delays for the rising and falling edges under worst-case conditions. Figures 6-9 and 6-10 apply to all 3.3V SL Enhanced Intel486 SX, DX, and DX2 CPUs. Figures 6-11 and 6-12 apply to 5V SL Enhanced Intel486 DX and DX2 CPUs. Figures 6-13 and 6-14 apply to 5V SL Enhanced Intel486 SX CPUs. The figures apply to all frequencies specified for each corresponding product.



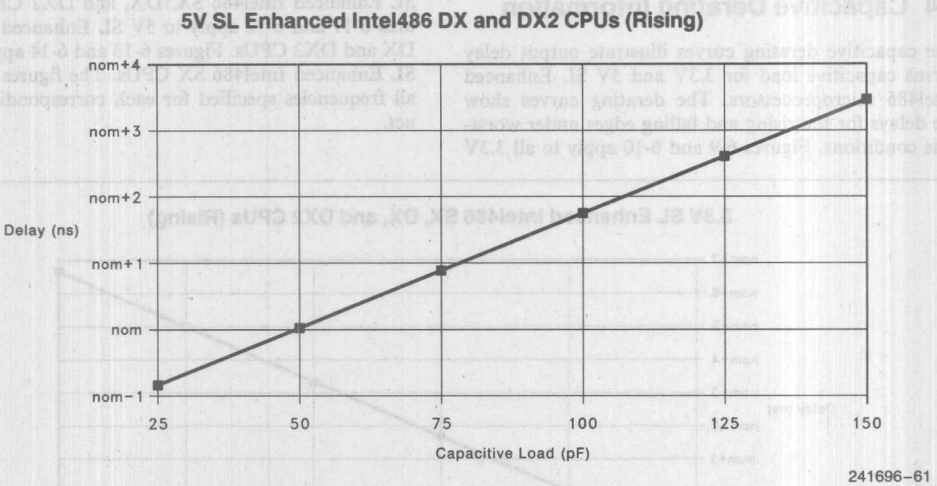
NOTE:
This graph will not be linear outside of the capacitive range shown.
nom = nominal value from the A.C. Characteristics table.

Figure 6-9. Typical Loading Delay versus Load Capacitance under Worst-Case Conditions for a Low-to-High Transition



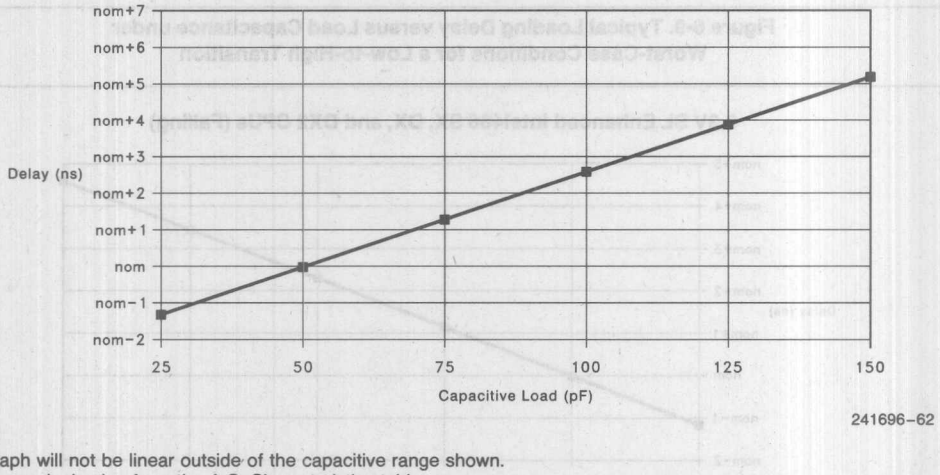
NOTE:
This graph will not be linear outside of the capacitive range shown.
nom = nominal value from the A.C. Characteristics table.

Figure 6-10. Typical Loading Delay versus Load Capacitance under Worst-Case Conditions for a High-to-Low Transition



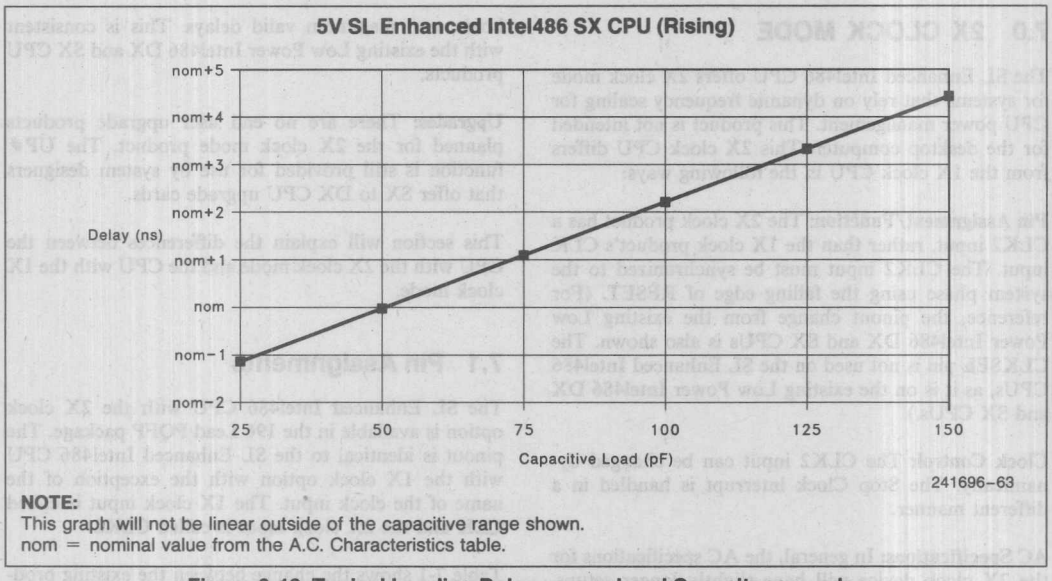
NOTE:
This graph will not be linear outside of the capacitive range shown.
nom = nominal value from the A.C. Characteristics table.

Figure 6-11. Typical Loading Delay versus Load Capacitance under Worst-Case Conditons for a Low-to-High Transition



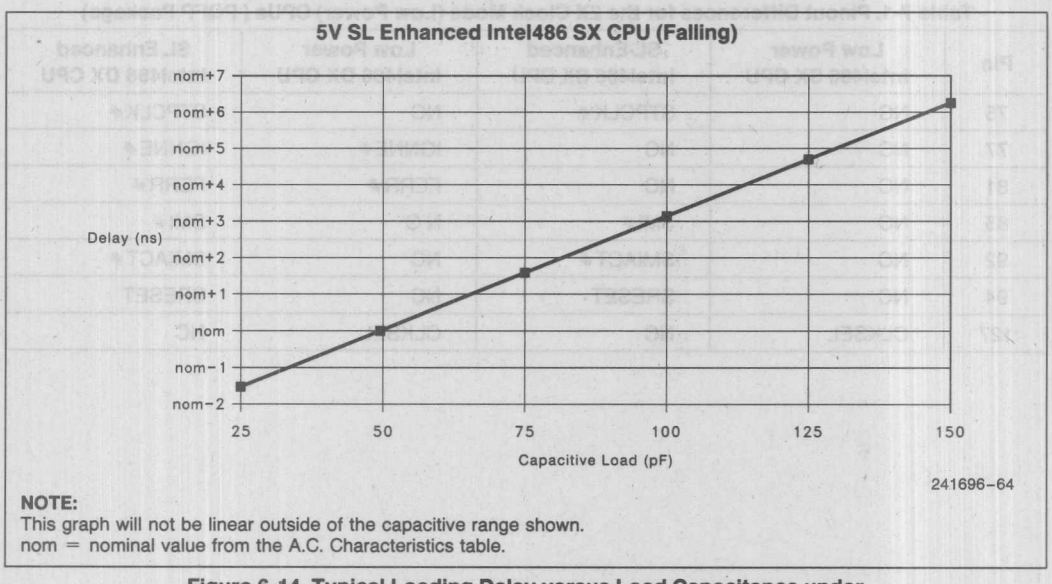
NOTE:
This graph will not be linear outside of the capacitive range shown.
nom = nominal value from the A.C. Characteristics table.

Figure 6-12. Typical Loading Delay versus Load Capacitance under Worst-Case Conditions for a High-to-Low Transition



NOTE:
This graph will not be linear outside of the capacitive range shown.
nom = nominal value from the A.C. Characteristics table.

Figure 6-13. Typical Loading Delay versus Load Capacitance under Worst-Case Conditions for a Low-to-High Transition



NOTE:
This graph will not be linear outside of the capacitive range shown.
nom = nominal value from the A.C. Characteristics table.

Figure 6-14. Typical Loading Delay versus Load Capacitance under Worst-Case Conditions for a High-to-Low Transition

7.0 2X CLOCK MODE

The SL Enhanced Intel486 CPU offers 2X clock mode for systems that rely on dynamic frequency scaling for CPU power management. This product is not intended for the desktop computer. This 2X clock CPU differs from the 1X clock CPU in the following ways:

Pin Assignment/Function: The 2X clock product has a CLK2 input, rather than the 1X clock product's CLK input. The CLK2 input must be synchronized to the system phase using the falling edge of RESET. (For reference, the pinout change from the existing Low Power Intel486 DX and SX CPUs is also shown. The CLKSEL pin is not used on the SL Enhanced Intel486 CPUs, as it is on the existing Low Power Intel486 DX and SX CPUs.)

Clock Control: The CLK2 input can be changed dynamically. The Stop Clock interrupt is handled in a different manner.

AC Specifications: In general, the AC specifications for the 2X clock device will have slightly longer setups, holds, and maximum valid delays. This is consistent with the existing Low Power Intel486 DX and SX CPU products.

Upgrades: There are no end user upgrade products planned for the 2X clock mode product. The UP# function is still provided for use by system designers that offer SX to DX CPU upgrade cards.

This section will explain the differences between the CPU with the 2X clock mode and the CPU with the 1X clock mode.

7.1 Pin Assignments

The SL Enhanced Intel486 CPU with the 2X clock option is available in the 196 Lead PQFP package. The pinout is identical to the SL Enhanced Intel486 CPU with the 1X clock option with the exception of the name of the clock input. The 1X clock input is called CLK and the 2X clock input is called CLK2.

Table 7-1 shows the change between the existing products and new products.

Table 7-1. Pinout Differences for the 2X Clock Mode (Low Power) CPUs (PQFP Package)

Pin	Low Power Intel486 SX CPU	SL Enhanced Intel486 SX CPU	Low Power Intel486 DX CPU	SL Enhanced Intel486 DX CPU
75	NC	STPCLK#	NC	STPCLK#
77	NC	NC	IGNNE#	IGNNE#
81	NC	NC	FERR#	FERR#
85	NC	SMI#	N C	SMI#
92	NC	SMIACT#	NC	SMIACT#
94	NC	SRESET	NC	SRESET
127	CLKSEL	NC	CLKSEL	NC

7.2 Quick Pin Reference

Table 7-2. Pin Descriptions

Symbol	Type	Name and Function
CLK2	I	**CLK2** provides the fundamental timing for the CPU. Both of the internal timing phases, phase-1 (ph1) and phase-2 (ph2), are provided by the external CLK2 input. All external timing parameters are specified with respect to the phase-1 rising edge of CLK2. For the 2X clock mode the CLK frequency is twice the frequency of the CPU.
RESET	I	The **RESET** input forces the CPU to begin execution at a known state. The CPU cannot begin execution of instructions until at least 1 ms after V_{CC} and CLK2 have reached their proper AC and DC specifications. However, for soft resets, RESET should remain active for at least 30 CLK2 periods (equal to 15 internal CPU CLK). The RESET pin should remain active during this time to insure proper CPU operation. RESET is active HIGH. Reset is asynchronous, but must meet setup and hold times t20, t20a and t21 for recognition in any specific clock. RESET sets the SMBASE descriptor to default address of 30000H. If the system uses SMBASE relocation, then the SRESET pin should be used for soft resets. For the 2X clock mode, the falling edge of RESET synchronizes the CPU internal clock phase. RESET must be used at power up and anytime the phase of the CPU clock must be re-synchronized to the system phase.
SRESET	I	The SRESET pin duplicates all the functionality of the RESET pin with the following two exceptions: 1. The SMBASE register will retain its previous value. 2. If UP# is asserted, SRESET will not have an effect on the host microprocessor. For soft resets, SRESET should remain active for at least 30 CLK2 periods (equal to 15 internal CPU CLK). SRESET is active HIGH. SRESET is asynchronous but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.

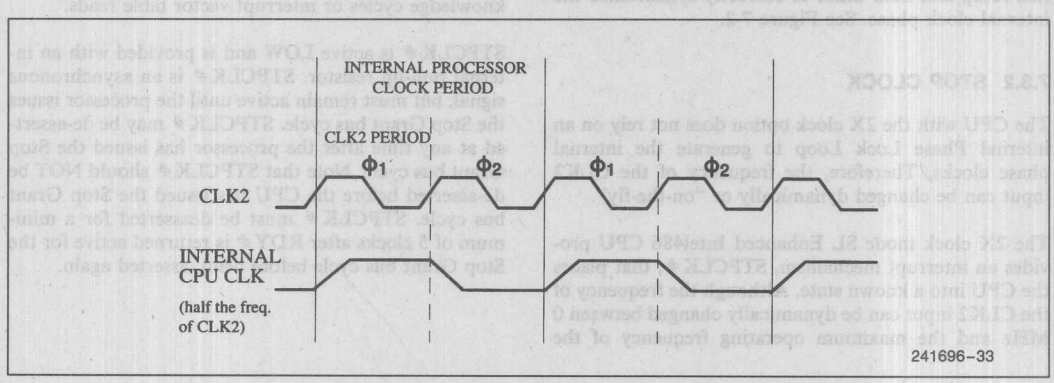


Figure 7-1. CLK2 Signal and Internal Processor Clock

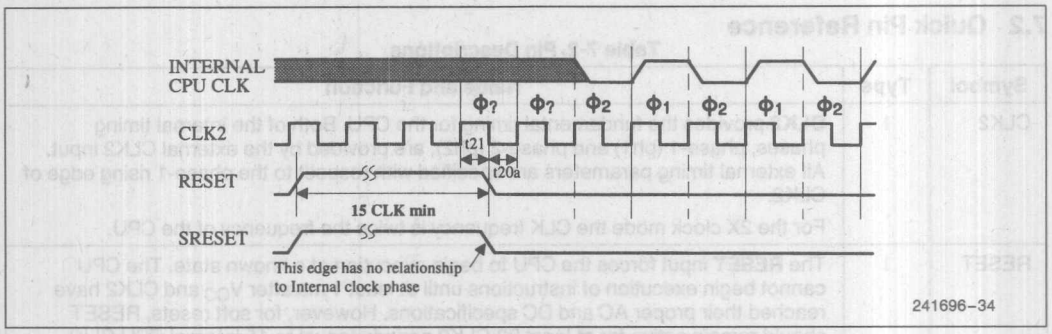


Figure 7-2. CLK2 and Internal Processor CLK vs. SRESET and RESET Timings

7.3 Clock Control

7.3.1 CLOCK GENERATION

The frequency of **CLK2** is twice the internal frequency of the CPU. The internal clock is comprised of two phases, "PH1" and "PH2". Each CLK2 period is a phase of the internal clock. Figure 7-1 illustrates the relationship between the CLK2 input and the internal phases. All set-up, hold, float-delay and valid delay timings are referenced to the rising edge of phase 1 of CLK2. Thus it is important to synchronize the external circuitry with the phase of the CLK2 input. The internal processor clock phase is determined at the falling edge of the RESET input. RESET must meet the specified setup and hold times to correctly synchronize the internal clock phase. See Figure 7-2.

7.3.2 STOP CLOCK

The CPU with the 2X clock option does not rely on an internal Phase Lock Loop to generate the internal phase clocks. Therefore, the frequency of the CLK2 input can be changed dynamically or "on-the-fly".

The 2X clock mode SL Enhanced Intel486 CPU provides an interrupt mechanism, STPCLK#, that places the CPU into a known state. Although the frequency of the CLK2 input can be dynamically changed between 0 MHz and the maximum operating frequency of the CPU, operation between 0 MHz and 8 MHz is not tested. Stopping the CLK2 input with the CPU in a known state requires use of the STPCLK# mechanism. When the CPU recognizes a STPCLK# request, the processor will stop execution on the next instruction boundary, stop the pre-fetcher, empty all internal pipelines and the write buffers, and then generate a Stop Grant bus cycle. At this point the CPU is in the Stop Grant state.

The rising edge of STPCLK# will tell the CPU that it can return to program execution at the instruction following the interrupted instruction.

Unlike the normal interrupts, INTR and NMI, the STPCLK# interrupt does not initiate interrupt acknowledge cycles or interrupt vector table reads.

STPCLK# is active LOW and is provided with an internal pull-up resistor. STPCLK# is an asynchronous signal, but must remain active until the processor issues the Stop Grant bus cycle. STPCLK# may be de-asserted at any time after the processor has issued the Stop Grant bus cycle. Note that STPCLK# should NOT be de-asserted before the CPU has issued the Stop Grant bus cycle. STPCLK# must be deasserted for a minimum of 5 clocks after RDY# is returned active for the Stop Grant bus cycle before being asserted again.

7.3.3 CLOCK CONTROL STATE DIAGRAM

The state diagram in Figure 7-3 shows the Stop Clock state transitions for the 2X clock mode.

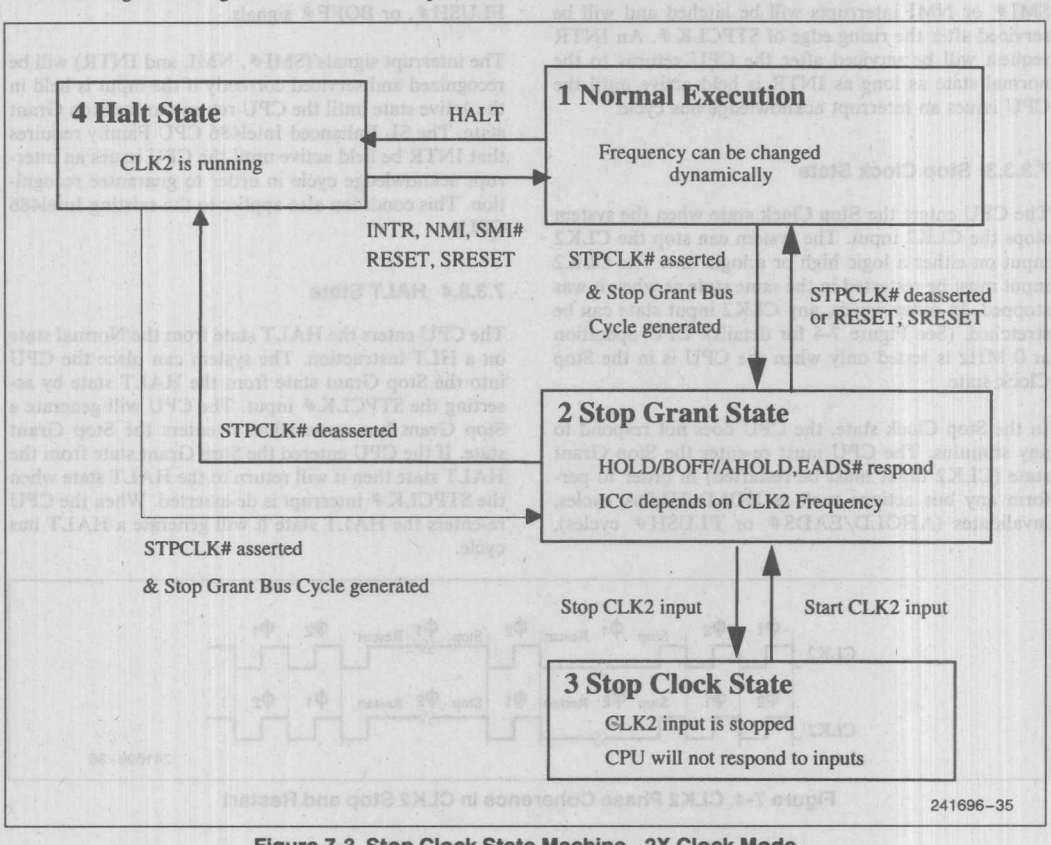


Figure 7-3. Stop Clock State Machine - 2X Clock Mode

7.3.3.1 Normal State

This is the Normal operating state of the CPU. During this state, the CLK2 input frequency can be changed dynamically or "on-the-fly" for power consumption control with no clock control latency. This capability provides a wide range of performance/power consumption options. Operation of the processor is tested between 8 MHz and the maximum operating frequency of the CPU. Operation below 8 MHz is guaranteed by design, though is not 100% tested. **Operation at 0 MHz is tested when the stop clock protocol (STPCLK#) is used.**

7.3.3.2 Stop Grant State

The CPU enters the **Stop Grant state** in response to a STPCLK# interrupt. The CPU will generate a Stop Grant bus cycle when it enters this state from the Normal state or the HALT state. The CPU will not generate a Stop Grant bus cycle when it enters the Stop Grant state from the Stop Clock state.

While in the **Stop Grant state**, the pull-up resistors on STPCLK# and UP# are disabled internally. The system must continue to drive these inputs to the state they were in immediately before the CPU entered the Stop Grant state. For minimum CPU power consumption, all other input pins should be driven to their inactive level while the CPU is in the Stop Grant state.

During the Stop Grant state, the CPU will respond to HOLD, AHOLD and BOFF# normally and can perform cache invalidates. An active edge on either the SMI# or NMI interrupts will be latched and will be serviced after the rising edge of STPCLK#. An INTR request will be serviced after the CPU returns to the normal state as long as INTR is held active until the CPU issues an interrupt acknowledge bus cycle.

7.3.3.3 Stop Clock State

The CPU enters the **Stop Clock state** when the system stops the CLK2 input. The system can stop the CLK2 input on either a logic high or a logic low. The CLK2 input must be restarted in the same state as when it was stopped. In other words, any CLK2 input state can be stretched. (See Figure 7-4 for details). CPU operation at 0 MHz is tested only when the CPU is in the Stop Clock state.

In the Stop Clock state, the CPU does not respond to any stimulus. The CPU must re-enter the Stop Grant state (CLK2 input must be restarted) in order to perform any bus actions such as HOLD/HLDA cycles, invalidates (AHOLD/EADS# or FLUSH# cycles), and BOFF#. It is recommended that CLK2 be restarted 2 clocks before and continue until 2 clocks after the transition of the HOLD, AHOLD, EADS#, FLUSH#, or BOFF# signals.

The interrupt signals (SMI#, NMI, and INTR) will be recognized and serviced correctly if the input is held in the active state until the CPU returns to the Stop Grant state. The SL Enhanced Intel486 CPU Family requires that INTR be held active until the CPU issues an interrupt acknowledge cycle in order to guarantee recognition. This condition also applies to the existing Intel486 CPUs.

7.3.3.4 HALT State

The CPU enters the HALT state from the Normal state on a HLT instruction. The system can place the CPU into the Stop Grant state from the HALT state by asserting the STPCLK# input. The CPU will generate a Stop Grant bus cycle when it enters the Stop Grant state. If the CPU entered the Stop Grant state from the HALT state then it will return to the HALT state when the STPCLK# interrupt is de-asserted. When the CPU re-enters the HALT state it will generate a HALT bus cycle.

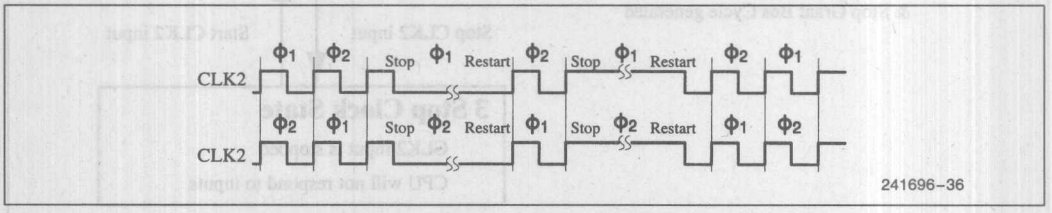


Figure 7-4. CLK2 Phase Coherence in CLK2 Stop and Restart

7.3.4 SUPPLY CURRENT MODEL FOR STOP CLOCK MODES AND TRANSITIONS

The following diagram illustrates the effect of different Stop Clock state transitions on the supply current.

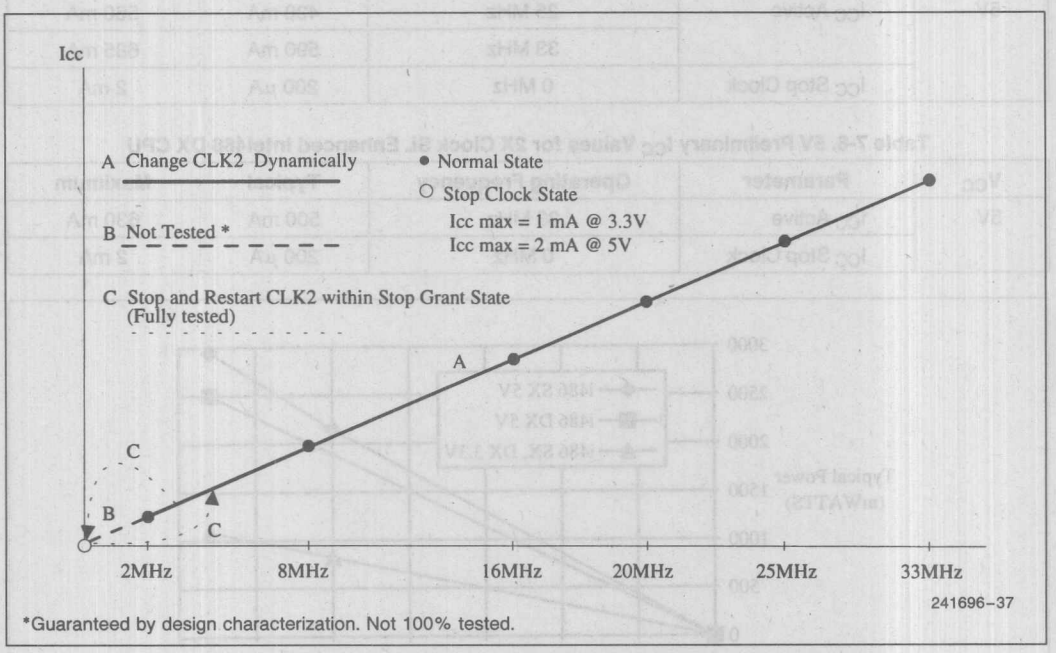


Figure 7-5. Supply Current Model for Stop Clock Modes and Transitions

Table 7-3. 3.3V Preliminary I_{CC} Values for 2X Clock SL Enhanced Intel486 SX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
3.3V	I_{CC} Active	25 MHz	250 mA	315 mA
		33 MHz	330 mA	415 mA
	I_{CC} Stop Clock	0 MHz	100 μA	1 mA

Table 7-4. 3.3V Preliminary I_{CC} Values for 2X Clock SL Enhanced Intel486 DX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
3.3V	I_{CC} Active	33 MHz	330 mA	415 mA
	I_{CC} Stop Clock	0 MHz	100 μA	1 mA

Table 7-5. 5V Preliminary I_{CC} Values for 2X Clock SL Enhanced Intel486™ SX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
5V	I_{CC} Active	25 MHz	430 mA	560 mA
		33 MHz	590 mA	685 mA
	I_{CC} Stop Clock	0 MHz	200 μA	2 mA

Table 7-6. 5V Preliminary I_{CC} Values for 2X Clock SL Enhanced Intel486 DX CPU

V_{CC}	Parameter	Operating Frequency	Typical	Maximum
5V	I_{CC} Active	33 MHz	500 mA	630 mA
	I_{CC} Stop Clock	0 MHz	200 μA	2 mA

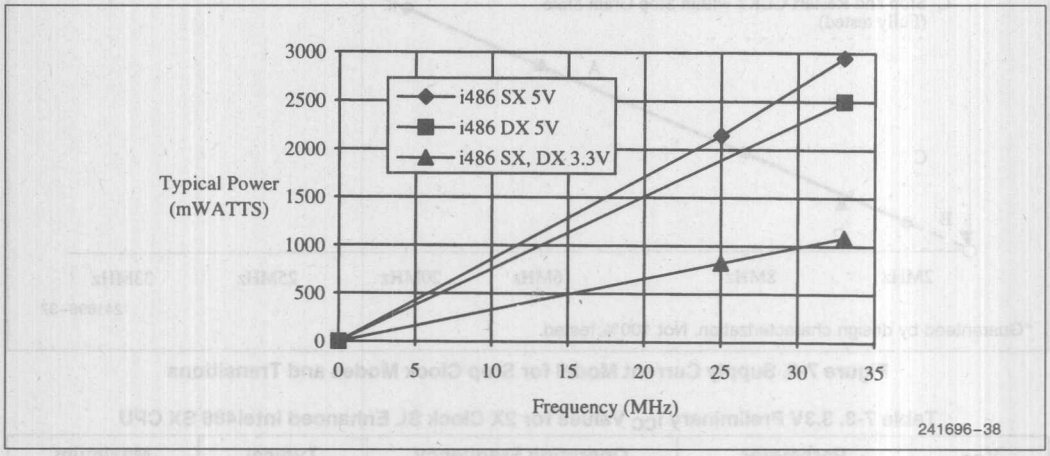


Figure 7-6. Frequency vs. Power (Typ) in 2X CLK Mode for 3.3V and 5V, SL Enhanced Intel486 SX and DX CPUs

7.4 D.C. Specifications for 2X Clock Option

For 2X clock D.C. specifications, refer to the 1X clock D.C. specifications (Section 6.1).

7.5 A.C. Specifications for 2X Clock Option

The A.C. specifications given in the tables of this section consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the rising edge of the phase 1 of the input system clock (CLK2), unless otherwise specified.

7.5.1 5V A.C. CHARACTERISTICS

Table 7-7. 5V A.C. Characteristics (2X Clock) Frequency = 25 and 33 MHz (Preliminary Information)
Functional operating range: V_{CC} = 5V ± 5%; T_{CASE} = 0 °C +85 °C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Min	Max	Unit	Notes
	Frequency		25		33	MHz	Note 1
	CLK2 Frequency		50		66	MHz	Note 1
t_1	CLK2 Period	20		15		ns	
t_2	CLK2 High Time	7		5		ns	at 2V
t_3	CLK2 Low Time	7		5		ns	at 0.8V
t_4	CLK2 Fall Time		2		2	ns	2V to 0.8V (2)
t_5	CLK2 Rise Time		2		2	ns	0.8V to 2V (2)
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA, SMIACT#, **FERR#** * Valid Delay	3	19	3	17	ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		28		21	ns	Note 2
t_8	PCHK# Valid Delay	3	24	3	23	ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	24	3	21	ns	
t_9	BLAST#, PLOCK# Float Delay		28		21	ns	Note 2
t_{10}	D0–D31, DP0–DP3 Write Data Valid Delay	3	20	3	19	ns	
t_{11}	D0–D31, DP0–DP3 Write Data Float Delay		28		21	ns	Note 2
t_{12}	EADS# Setup Time	9		6		ns	
t_{13}	EADS# Hold Time	4		4		ns	
t_{14}	KEN#, BS16#, BS8# Setup Time	9		6		ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	4		4		ns	
t_{16}	RDY#, BRDY# Setup Time	9		6		ns	
t_{17}	RDY#, BRDY# Hold Time	4		4		ns	
t_{18}	HOLD, AHOLD Setup Time	11		7		ns	
t_{18a}	BOFF# Setup Time	11		9		ns	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	4		4		ns	
t_{20}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#** * Setup Time	11		6		ns	
t_{20a}	RESET Falling Edge Setup Time	9		5		ns	
t_{21}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#** * Hold Time	4		4		ns	

Table 7-7. 5V A.C. Characteristics (2X Clock) Frequency = 25 and 33 MHz (Preliminary Information) (Continued)

Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Min	Max	Unit	Notes
t_{22}	D0–D31, DP0–DP3, A4–A31 Read Setup Time	6		6		ns	
t_{23}	D0–D31, DP0–DP3, A4–A31 Read Hold Time	4		4		ns	

NOTES:
* Present only in the SL Enhanced Intel486™ DX microprocessor.
1. 0 MHz operation is tested using the STPCLK# and STOP GRANT bus cycle protocol. Operation between 0 MHz < CLK2 < 8 MHz is guaranteed by design characterization, but is not 100% tested.
2. Not 100% tested, guaranteed by design characterization.

7.5.2 3.3V A.C. CHARACTERISTICS

Table 7-8. 3.3V A.C. Characteristics (2X Clock) Frequency = 25 and 33 MHz (Preliminary Information)

Functional operating range: V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Min	Max	Unit	Notes
	Frequency		**25**		**33**	MHz	Note 1
	CLK2 Frequency		**50**		**66**	MHz	Note 1
t_1	CLK2 Period	20		15		ns	
t_2	CLK2 High Time	7		5		ns	at 2V
t_3	CLK2 Low Time	7		5		ns	at 0.8V
t_4	CLK2 Fall Time		2		2	ns	2V to 0.8V(2)
t_5	CLK2 Rise Time		2		2	ns	0.8V to 2V(2)
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA, SMIACT#, **FERR#*** Valid Delay	3	19	3	17	ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		28		21	ns	Note 2
t_8	PCHK# Valid Delay	3	24	3	23	ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	24	3	21	ns	
t_9	BLAST#, PLOCK# Float Delay		28		21	ns	Note 2
t_{10}	D0–D31, DP0–DP3 Write Data Valid Delay	3	20	3	19	ns	
t_{11}	D0–D31, DP0–DP3 Write Data Float Delay		28		21	ns	Note 2
t_{12}	EADS# Setup Time	9		6		ns	
t_{13}	EADS# Hold Time	4		4		ns	
t_{14}	KEN#, BS16#, BS8# Setup Time	9		6		ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	4		4		ns	
t_{16}	RDY#, BRDY# Setup Time	9		6		ns	
t_{17}	RDY#, BRDY# Hold Time	4		4		ns	
t_{18}	HOLD, AHOLD Setup Time	11		7		ns	
t_{18a}	BOFF# Setup Time	11		9		ns	

Table 7-8. 3.3V A.C. Characteristics (2X Clock) Frequency = 25 and 33 MHz (Preliminary Information) (Continued)

Functional operating range: V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Min	Max	Unit	Notes
t_{19}	HOLD, AHOLD, BOFF# Hold Time	4		4		ns	
t_{20}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#*** Setup Time	11		6		ns	
t_{20a}	RESET Falling Egde Setup Time	9		5		ns	
t_{21}	FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, RESET, **IGNNE#*** Hold Time	4		4		ns	
t_{22}	D0–D31, DP0–DP3, A4–A31 Read Setup Time	6		6		ns	
t_{23}	D0–D31, DP0–DP3, A4–A31 Read Hold Time	4		4		ns	

NOTES:
* Present only in the SL Enhanced Intel486™ DX Microprocessor
1. 0 MHz operation is tested using the STPCLK# and STOP GRANT bus cycle protocol. Operation between 0 MHz<CLK2<8 MHz is guaranteed by design characterization, but is not 100% tested.
2. Not 100% tested, guaranteed by design characterization.

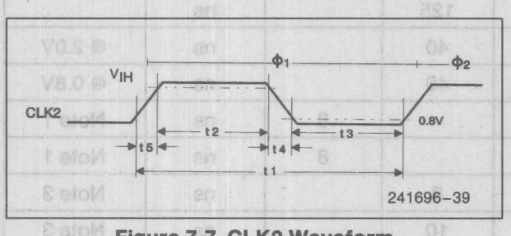


Figure 7-7. CLK2 Waveform

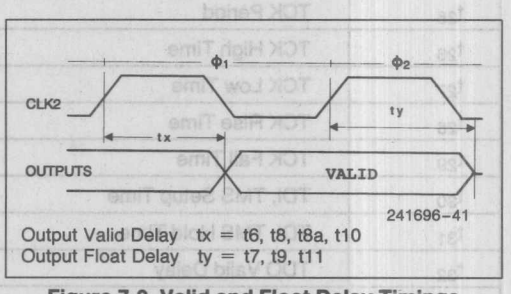


Figure 7-9. Valid and Float Delay Timings

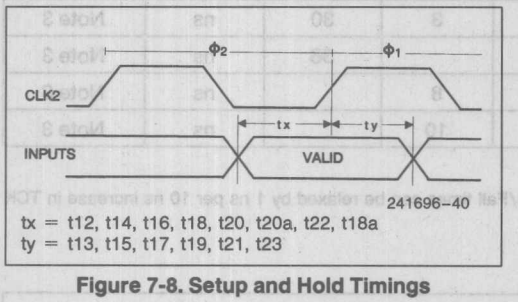


Figure 7-8. Setup and Hold Timings

8.0 TESTABILITY

8.1 Test Access Port

TCK
This Test Clock signal is an input to the CPU and provides the clocking function required by the JTAG Boundary scan feature. TCK is used to clock state information and data into the component on the rising edge of TCK on TMS and TDI, respectively. Data is clocked out of the CPU on the falling edge of TCK and TDO.

TDI
This Test Data Input signal is the serial input used to shift JTAG instruction and data into the CPU. TDI is sampled on the rising edge of TCK, during the SHIFT-IR and SHIFT-DR TAP controller states. During all other tap controller states, TDI is a "don't care".

TDO
This Test Data Output signal is the serial output used to shift JTAG instructions and the data out of the CPU. TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT-DR TAP controller states. At all other times, TDO is driven to the high impedance state.

TMS
This Test Mode Select signal is decoded by the JTAG TAP (Test Access Port) to select the operation of the test logic. TMS is sampled on the rising edge of TCK. To guarantee deterministic behavior of the TAP controller, TMS is provided with an internal pull-up resistor.

Table 8-1. A.C. Specifications for the Test Access Port

Symbol	Parameter	Min	Max	Unit	Notes
t_{24}	TCK Frequency		8	MHz	Note 2
t_{25}	TCK Period	125		ns	
t_{26}	TCK High Time	40		ns	@ 2.0V
t_{27}	TCK Low Time	40		ns	@ 0.8V
t_{28}	TCK Rise Time		8	ns	Note 1
t_{29}	TCK Fall Time		8	ns	Note 1
t_{30}	TDI, TMS Setup Time	8		ns	Note 3
t_{31}	TDI, TMS Hold Time	10		ns	Note 3
t_{32}	TDO Valid Delay	3	30	ns	Note 3
t_{33}	TDO Float Delay		36	ns	Note 3
t_{34}	All outputs (Non-Test Valid Delay)	3	30	ns	Note 3
t_{35}	All Outputs (Non-Test Float Delay)		36	ns	Note 3
t_{36}	All Inputs (Non-Test) Setup Time	8		ns	Note 3
t_{37}	All Inputs (Non-Test) Hold Time	10		ns	Note 3

NOTES:
1. Rise/Fall Times are measured between 0.8V and 0.2V. Rise/Fall times can be relaxed by 1 ns per 10 ns increase in TCK period.
2. TCK period ≥ CLK period.
3. Parameter measured from TCK.

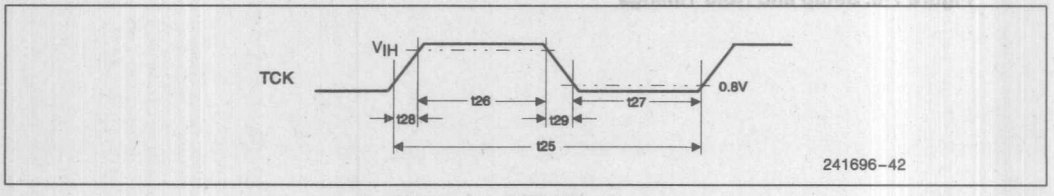


Figure 8-1. TCK Waveform

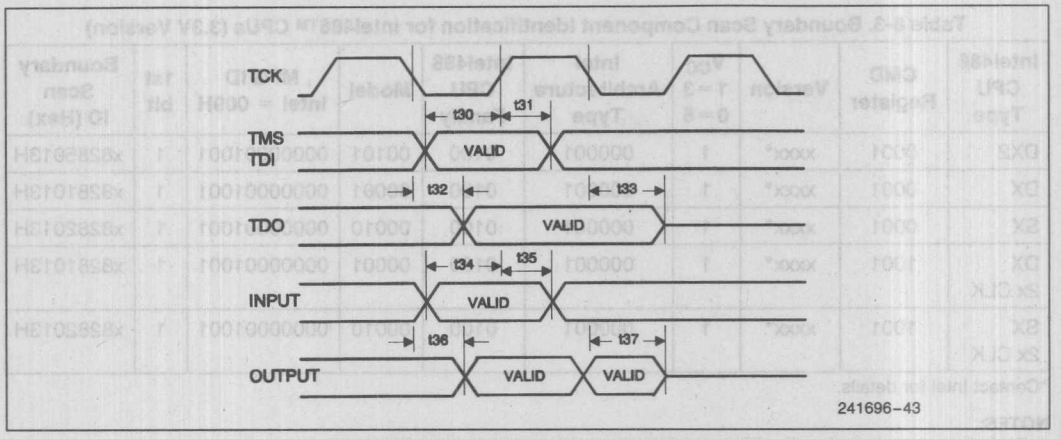


Figure 8-2. Test Signal Timing Diagram

8.2 Boundary Scan Component Identification Support

The IEEE standard 1149.1 includes an optional public instruction called "IDCODE" which is used to verify the correct parts are installed in the correct locations on a board. The SL Enhanced Intel486 CPUs support this instruction with a IDCODE value of 0010. When implemented IEEE requires that the register that is selected for the scan path be 32 bits long with four specific sub fields. These fields are:

- LSB (bit 0) must have a value of "1".
- Manufacturer Identity (bits 1–11). This is a compressed form of the JEDEC (pub 106-A) manufacturers code. Intel is identified by a hex value of 009.
- Part Number (bits 12–27). IEEE only requires that 2 different parts in the same package style with the TAP pins in the same location must have different values. Intel has sub divided this field further as indicated in the following tables. These sub fields are V_{CC}, Type, Family, and Model.
- Version (bits 28–31). IEEE recommends that this field should be assigned to identify the variant of a component type. Intel is using this field to identify major steppings of the parts.

Table 8-2. Boundary Scan Component Identification for Intel486™ CPUs (5V Versions)

Intel486 CPU Type	CMD Register	Version	V_{CC} 1=3 0=5	Intel Architecture Type	Intel486 CPU Family	Model	MFG ID Intel = 009H	1st bit	Boundary Scan ID (Hex)
DX2	0001	xxxx*	0	000001	0100	00101	00000001001	1	x0285013H
DX	0001	xxxx*	0	000001	0100	00001	00000001001	1	x0281013H
SX	0001	xxxx*	0	000001	0100	00010	00000001001	1	x0282013H
DX 2x CLK	1001	xxxx*	0	000001	0100	00001	00000001001	1	x0281013H
SX 2x CLK	1001	xxxx*	0	000001	0100	00010	00000001001	1	x0282013H

*Contact Intel for details.

Table 8-3. Boundary Scan Component Identification for Intel486™ CPUs (3.3V Version)

Intel486 CPU Type	CMD Register	Version	V_{CC} 1=3 0=5	Intel Architecture Type	Intel486 CPU Family	Model	MFG ID Intel = 009H	1st bit	Boundary Scan ID (Hex)
DX2	0001	xxxx*	1	000001	0100	00101	00000001001	1	x8285013H
DX	0001	xxxx*	1	000001	0100	00001	00000001001	1	x8281013H
SX	0001	xxxx*	1	000001	0100	00010	00000001001	1	x8282013H
DX 2x CLK	1001	xxxx*	1	000001	0100	00001	00000001001	1	x8281013H
SX 2x CLK	1001	xxxx*	1	000001	0100	00010	00000001001	1	x8282013H

*Contact Intel for details.

NOTES:
1. DX indicates the inclusion of a FPU, and SX indicates the exclusion of a FPU.
2. The distinction between 1x and 2x bus clocks is determined by reading the 4 bits loaded into the shift-register during the "Capture-IR" state. (labeled here as "cmd reg")

9.0 OverDrive™ PROCESSOR SOCKET FOR SL ENHANCED INTEL486 CPU-BASED SYSTEMS

This section provides OverDrive Processor socket specifications for systems based on the SL Enhanced Intel486 SX, DX and DX2 CPUs.

The OverDrive Processor socket for Intel486 DX2 CPU-based systems is a superset of the OverDrive Processor socket for Intel486 SX and DX CPU-based systems. Section 9.1 specifies the OverDrive Processor socket for Intel486 SX and DX CPU-based systems. Section 9.2 specifies the Pentium™ OverDrive Processor socket for 5V Intel486 DX2 CPU-based systems.

The OverDrive Processor socket specifications in this section are only valid for systems using 1X clock mode CPUs. There are no OverDrive Processor socket specifications compatible with 2X clock mode CPUs.

9.1 OverDrive Processor Socket for 5V SL Enhanced Intel486 SX and DX CPU-Based Systems

This section contains OverDrive Processor socket specifications for 5V systems based on the SL Enhanced Intel486 SX or DX CPUs. The OverDrive Processor socket specifications are also valid for 5V systems based on the existing Intel486 SX or DX CPUs.

9.1.1 OVERDRIVE PROCESSOR SOCKET CIRCUIT DESIGN

Figures 9-1 and 9-2 show the circuits which interface the original CPU with the OverDrive Processor socket. These circuits allow SL Enhanced Intel486 SX and DX (as well as Intel486 SX and DX) CPU-based systems to be upgraded with the OverDrive Processor.

9.1.1.1 OverDrive Processor Socket Circuit for SL Enhanced Intel486 DX CPU-Based Systems

The OverDrive Processor socket circuit for Intel486 DX CPU-based systems allows the Intel486 DX CPU to directly recognize when the OverDrive Processor socket is populated. When the UP# pin is driven active to the Intel486 DX CPU, the CPU three-states all of its output pins and enters power-down mode. This circuit is shown in Figure 9-1.

9.1.1.2 OverDrive Processor Socket Circuit for SL Enhanced Intel486 SX CPU-Based Systems

The OverDrive Processor socket circuit for Intel486 SX CPU-based systems allows the microprocessor to directly recognize when the OverDrive Processor socket is populated. When the UP# pin is driven active to the Intel486 SX CPU, the CPU three-states all of its output pins and enters power-down mode. This circuit is shown in Figure 9-2.

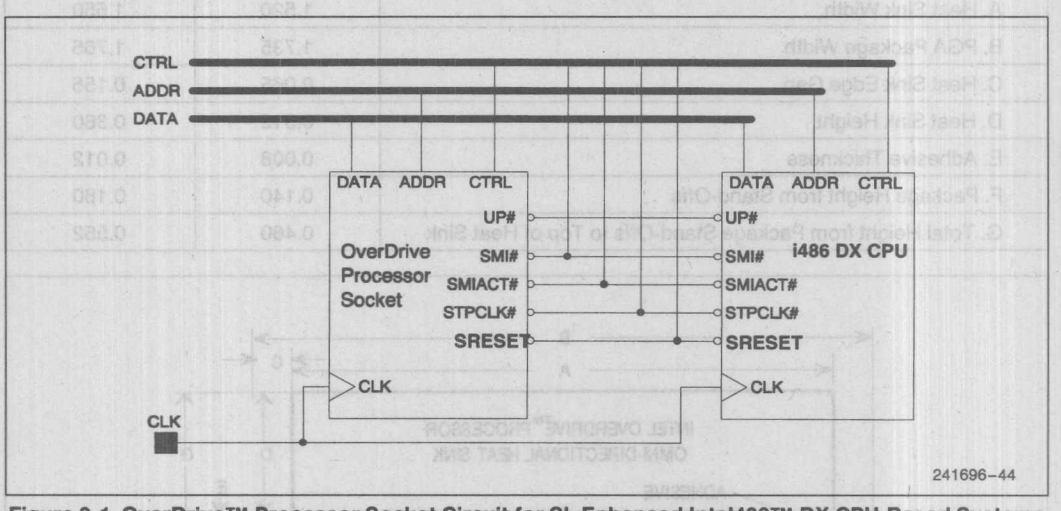


Figure 9-1. OverDrive™ Processor Socket Circuit for SL Enhanced Intel486™ DX CPU-Based Systems

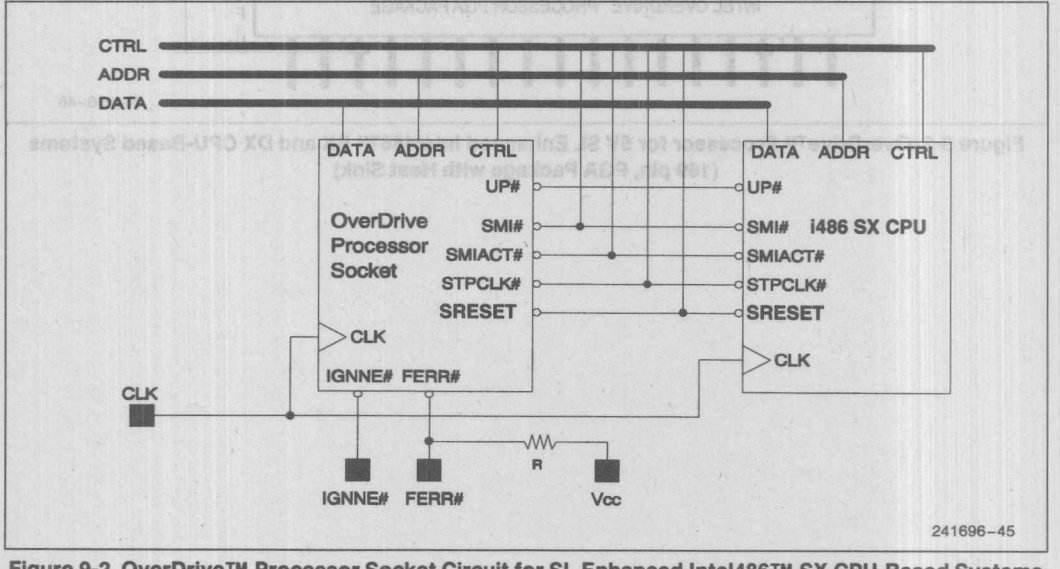


Figure 9-2. OverDrive™ Processor Socket Circuit for SL Enhanced Intel486™ SX CPU-Based Systems

9.1.2 SOCKET LAYOUT

The OverDrive Processors for Intel486 SX and DX CPU-based systems are supplied with an attached heat sink. Intel486 SX and DX microprocessor system designs must provide space for the heat sink on the OverDrive Processor.

The maximum and minimum dimensions for the 169 pin, PGA package with heat sink are shown in Table 9-1. The maximum height of the OverDrive Processor for 5V Intel486 SX and DX CPU-based systems from the pin stand-offs to the top of the heat sink, including the adhesive thickness, is 0.552 inches.

Table 9-1 OverDrive™ Processor, 169 pin, PGA Package Dimensions With Heat Sink Attached

DIMENSION (inches)	Minimum	Maximum
A. Heat Sink Width	1.520	1.550
B. PGA Package Width	1.735	1.765
C. Heat Sink Edge Gap	0.065	0.155
D. Heat Sink Height	0.312	0.360
E. Adhesive Thickness	0.008	0.012
F. Package Height from Stand-Offs	0.140	0.180
G. Total Height from Package Stand-Offs to Top of Heat Sink	0.460	0.552

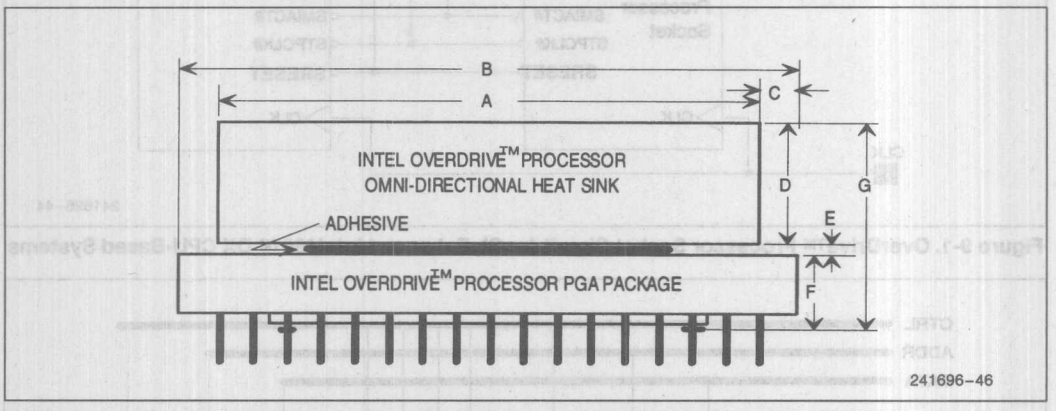


Figure 9-3. OverDrive™ Processor for 5V SL Enhanced Intel486™ SX and DX CPU-Based Systems (169 pin, PGA Package with Heat Sink)

9.1.3 THERMAL MANAGEMENT

The heat generated by the OverDrive Processor requires careful management of heat dissipation.

Thermal Equations: The method for calculating the heat dissipation for an OverDrive Processor (with a heat sink) or a standard CPU is the same except that the reference point for measuring the device temperature is different. The OverDrive Processor specifies T_{sink} (the temperature at the outside center base of the heat sink, not on the heat sink marking plate or cooling posts) and θ_{JS} (the thermal resistance from the silicon junction to the heat sink base). The relationships between temperature, thermal resistance and power are shown in the following equations:

$$T_{sink} = T_{ambient} + (P_{max} * \theta_{SA})$$

where,

$T_{ambient}$ = Ambient Temperature,
P_{max} = Power ($I_{CC} * V_{CC}$),
θ_{SA} = $\theta_{JA} - \theta_{JS}$.

The maximum I_{CC} values in Table 9-2 are the best known design estimates and should be used as the maximum specification.

Table 9-2. OverDrive™ Processor Maximum and Typical I_{CC} Values for 5V Intel486 SX and DX CPU-Based Sytems

System Frequency (MHz)	Typical I_{CC} (mA)	Maximum I_{CC} (mA)
25	TBD	1000
33	TBD	1250

The thermal resistance values for the OverDrive Processor (169 pin, PGA package with heat sink) are shown in Table 9-3. The maximum TA values shown in Table 9-4 were calculated using T_S = 85°C, V_{CC} = 5.3V, the maximum I_{CC} values shown in Table 9-2, and θ_{JS} and θ_{JA} values shown in Table 9-3.

Table 9-3. Thermal Resistance for the OverDrive Processor for 5V Intel486 SX and DX CPU-Based Systems

5V OverDrive Processor	θ_{JS}	Airflow (Ft/min., LFM)				
	1.9°C/W	0	200	400	600	800
θ_{JA} (°C/W)		12.0	8.6	6.6	5.1	4.6

Table 9-4. Maximum T_A for the OverDrive™ Processor for 5V Intel486 SX and DX CPU-Based Systems

5V OverDrive Processor	Airflow (Ft/min., LFM)					
	f_{CLK}(MHz)	0	200	400	600	800
T_A(°C)	25	31	49	60	68	70
	33	18	40	53	63	67

For further details about thermal and mechanical package specifications and methodologies, refer to the 1993 Packaging Handbook (order number 240800).

9.1.4 DESIGN CONSIDERATIONS

Timing Dependent Loops

The OverDrive Processor may internally execute instructions at multiple frequencies of the input clock. Thus software (or instruction based) timing loops will execute faster on the OverDrive Processor than on the Intel486 SX or DX CPU (at the same input clock frequency). Instructions such as NOP, LOOP, and JMP $+2, have been used by BIOS to implement timing loops, that are required, for example, to enforce recovery time between consecutive accesses for I/O devices. These instruction-based timing loop implementations may require modification for systems intended to be upgraded with the OverDrive Processor.

In order to avoid any incompatibilities, it is recommended that timing loops be implemented in hardware rather than in software. This provides transparency and also requires no change to BIOS or I/O device drivers in the future when moving to higher processor clock speeds. As an example, a timing loop may be implemented as follows: The software performs a dummy I/O instruction to an unused I/O port. The hardware for the bus controller logic recognizes this I/O instruc-

tion and delays the termination of the I/O cycle to the CPU by keeping RDY# or BRDY# deasserted for the appropriate amount of time.

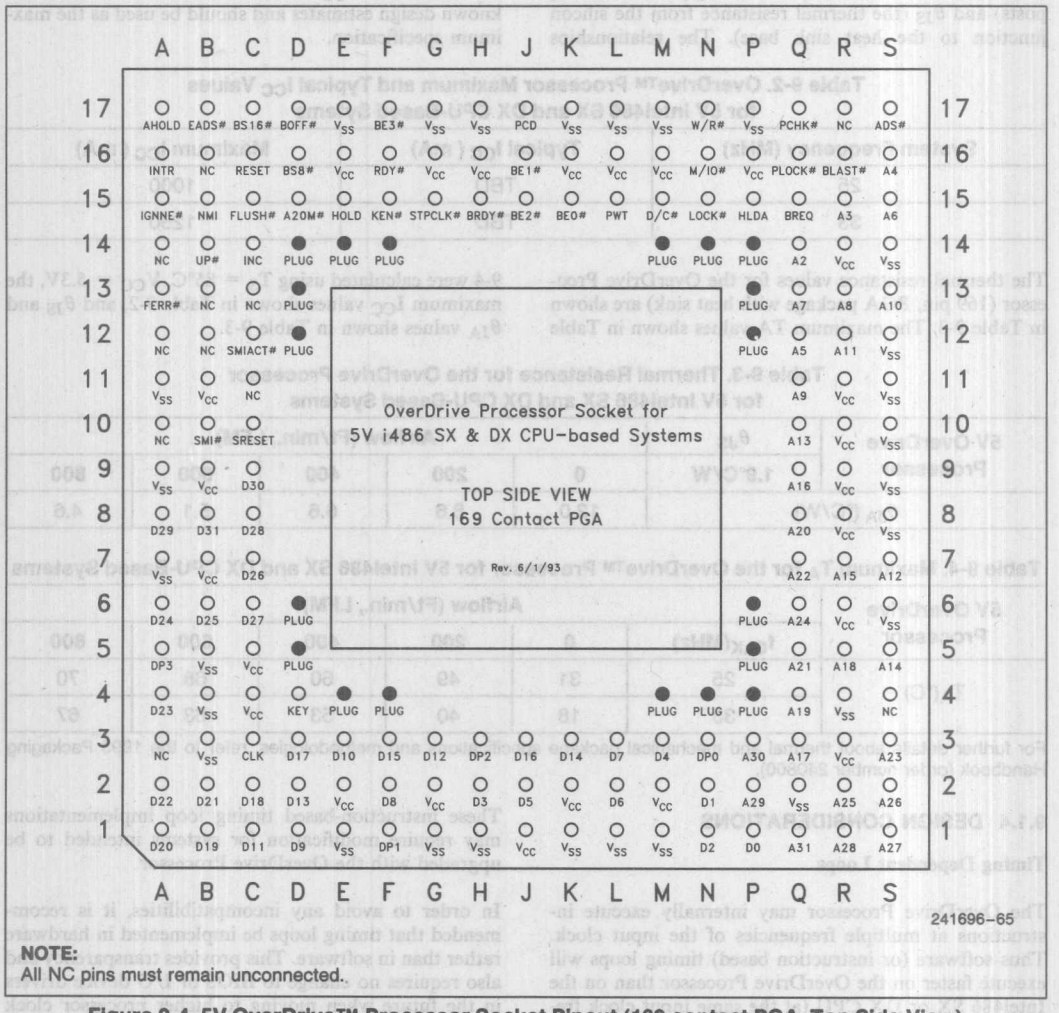
Boundary Scan

The OverDrive Processor does not support the Boundary Scan feature.

9.1.5 TESTABILITY

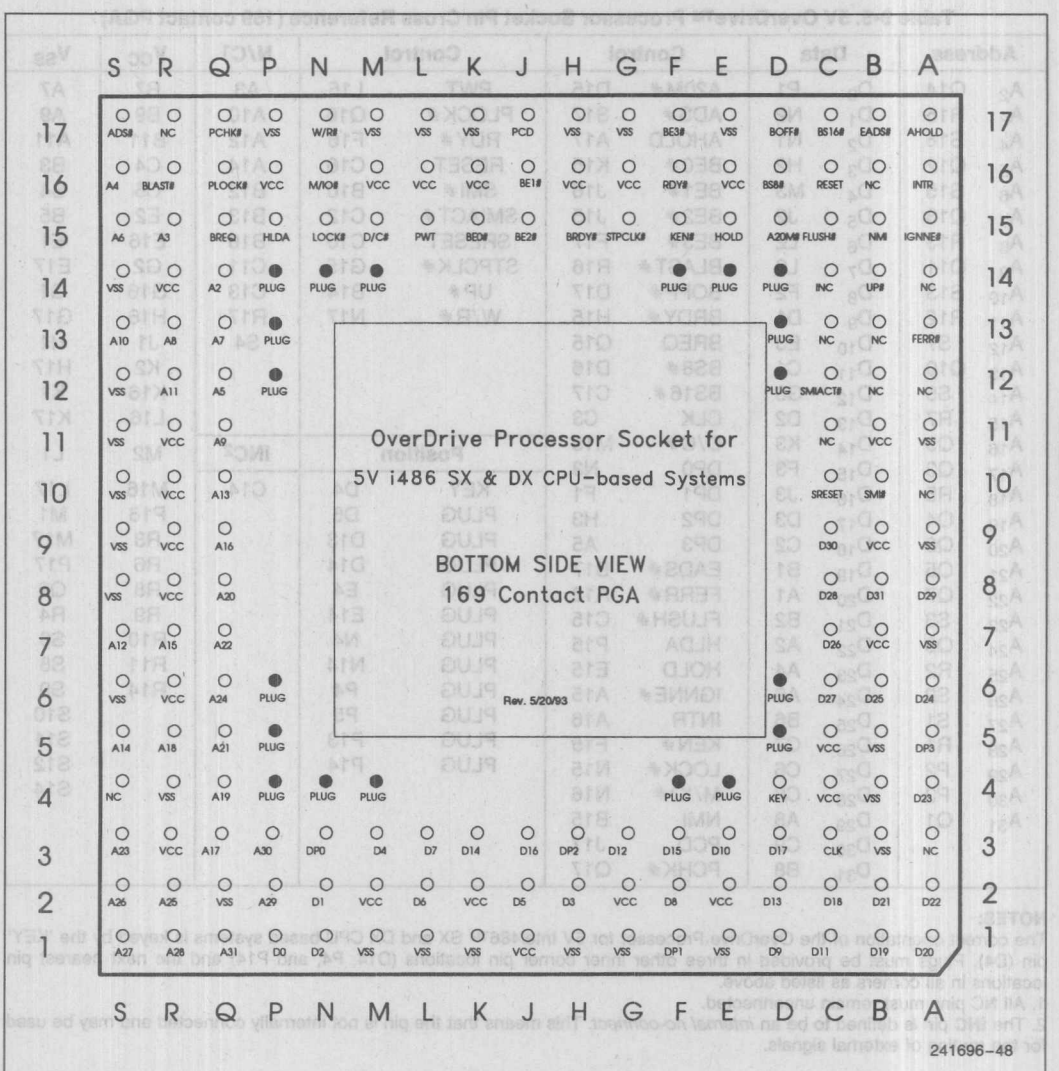
The electrical functionality of the OverDrive Processor socket can be verified by fully testing the system with a populated OverDrive Processor socket. We recommend the system be tested with all compatible OverDrive Processors to ensure there are no compatibility issues.

9.1.6 OVERDRIVE PROCESSOR SOCKET PINOUT FOR 5V INTEL486 SX AND DX CPU-BASED SYSTEMS



NOTE:
All NC pins must remain unconnected.

Figure 9-4. 5V OverDrive™ Processor Socket Pinout (169 contact PGA, Top Side View)



NOTE:
All NC pins must remain unconnected.

Figure 9-5. 5V OverDrive™ Processor Socket Pinout (169 contact PGA, Bottom Side View)

Table 9-5. 5V OverDrive™ Processor Socket Pin Cross Reference (169 contact PGA)

Address		Data		Control		Control		N/C[1]	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	PWT	L15	A3	B7	A7
A_3	R15	D_1	N2	ADS#	S17	PLOCK#	Q16	A10	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	RDY#	F16	A12	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	RESET	C16	A14	C4	B3
A_6	S15	D_4	M3	BE1#	J16	SMI#	B10	B12	C5	B4
A_7	Q13	D_5	J2	BE2#	J15	SMIACT#	C12	B13	E2	B5
A_8	R13	D_6	L2	BE3#	F17	SRESET	C10	B16	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16	STPCLK#	G15	C11	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17	UP#	B14	C13	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15	W/R#	N17	R17	H16	G17
A_{12}	S7	D_{10}	E3	BREQ	Q15			S4	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16				K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17				K16	K1
A_{15}	R7	D_{13}	D2	CLK	C3				L16	K17
A_{16}	Q9	D_{14}	K3	D/C#	M15	**Position**		**INC[2]**	M2	L1
A_{17}	Q3	D_{15}	F3	DP0	N3					
A_{18}	R5	D_{16}	J3	DP1	F1	KEY	D4	C14	M16	L17
A_{19}	Q4	D_{17}	D3	DP2	H3	PLUG	D5		P16	M1
A_{20}	Q8	D_{18}	C2	DP3	A5	PLUG	D13		R3	M17
A_{21}	Q5	D_{19}	B1	EADS#	B17	PLUG	D14		R6	P17
A_{22}	Q7	D_{20}	A1	FERR#	A13	PLUG	E4		R8	Q2
A_{23}	S3	D_{21}	B2	FLUSH#	C15	PLUG	E14		R9	R4
A_{24}	Q6	D_{22}	A2	HLDA	P15	PLUG	N4		R10	S6
A_{25}	R2	D_{23}	A4	HOLD	E15	PLUG	N14		R11	S8
A_{26}	S2	D_{24}	A6	IGNNE#	A15	PLUG	P4		R14	S9
A_{27}	S1	D_{25}	B6	INTR	A16	PLUG	P5			S10
A_{28}	R1	D_{26}	C7	KEN#	F15	PLUG	P13			S11
A_{29}	P2	D_{27}	C6	LOCK#	N15	PLUG	P14			S12
A_{30}	P3	D_{28}	C8	M/IO#	N16					S14
A_{31}	Q1	D_{29}	A8	NMI	B15					
		D_{30}	C9	PCD	J17					
		D_{31}	B8	PCHK#	Q17					

NOTES:
The correct orientation of the OverDrive Processor for 5V Intel486™ SX and DX CPU-based systems is keyed by the 'KEY' pin (D4). Plugs must be provided in three other inner corner pin locations (D14, P4, and P14) and the next nearest pin locations in all corners as listed above.
1. All NC pins must remain unconnected.
2. The **INC** pin is defined to be an *internal no-connect*. This means that the pin is not internally connected and may be used for the routing of external signals.

9.1.7 D.C./A.C. CHARACTERISTICS

The OverDrive Processor is compatible with the maximum rating specification and the D.C./A.C. specifications of the Intel486 SX and DX CPUs. Refer to the thermal management section (Section 9.1.3) for the maximum power supply current and operating temperature specifications.

9.2 Pentium™ OverDrive Processor Socket for 5V SL Enhanced Intel486 DX2 CPU-Based Systems

This section contains the specification for the Pentium OverDrive Processor socket for systems based on the Intel486 DX2 CPU. These OverDrive Processor socket specifications are also valid for 5V systems based on the existing Intel486 DX2 CPU. Please note that the Pentium OverDrive Processor socket specifications presented in this document are preliminary and subject to change. For more updated information about the OverDrive Processor socket, please contact your local Intel representative.

9.2.1 PENTIUM OVERDRIVE PROCESSOR SOCKET OVERVIEW FOR 5V SL ENHANCED Intel486 DX2 CPU-BASED SYSTEMS

The OverDrive Processor socket specifies 237 contacts, which correspond to a standard 240 pin socket with one inside "KEY" contact, one outer "KEY" contact and four "orientation" contacts plugged on the outside corner. The inside "KEY" contact provides backward compatibility with the OverDrive Processor socket for Intel486 SX and DX CPU-based systems. The four contacts plugged on the outside corner and the outer "KEY" pin ensure proper orientation for the Pentium OverDrive Processor. Additionally, all 120 inner contacts of a standard 240 pin socket (the inner 11 rows minus the "KEY" pin) should be plugged to ensure proper alignment of the OverDrive Processor for Intel486 SX and DX CPU-based systems.

To support Intel's SMM of the SL Enhanced Intel486 processor, new pins have been added to the Pentium OverDrive Processor socket.

The **INC** Pin is defined to be an internal no-connect. This means that the pin is not internally connected and may be used for the routing of external signals. For example, pin D15 is defined to be the **FERR#** pin on the Intel486 DX CPU. By routing D15 and B14 together, the **FERR#** signal can be made to appear on both pins.

Note that the **SRESET** pin has been defined at location D11 for compatibility with the SL Enhanced Intel486 DX2 Processor. On the Pentium OverDrive processor, D11 is defined as an **INC** pin. The OverDrive processor signal **INIT** and the SL Enhanced Intel486 CPU signal **SRESET** may be electrically connected on the motherboard, provided the system can handle the functionality differences depending on which part is inserted into the socket. Specifically, **INIT** and **SRESET** are similar except that **SRESET** will flush the on-chip cache, while **INIT** will keep the cache contents intact. Systems should not depend on **SRESET** to flush the cache in future versions of the SL Enhanced Intel486 processor, as previously described in this document.

9.2.2 MECHANICAL DESIGN CONSIDERATIONS

The Pentium OverDrive Processor is designed to fit in a standard 240-lead (19 x 19) PGA socket with four corner pins removed. The Pentium OverDrive Processor uses an active heat sink, and therefore, requires vertical clearance to allow adequate air circulation.

The maximum and minimum dimensions of the Pentium OverDrive Processor package with a fan/heat sink are shown in Table 9-6. The fan/heat sink unit is divided into the size of the actual heat sink, and the required free space above the heat sink. The total height required for the Pentium OverDrive Processor from the motherboard will depend on the height of the PGA socket. The total external height given in Table 9-6 is only measured from the PGA pin stand-offs. Table 9-6 also details the minimum clearance needed around all four sides of the PGA package.

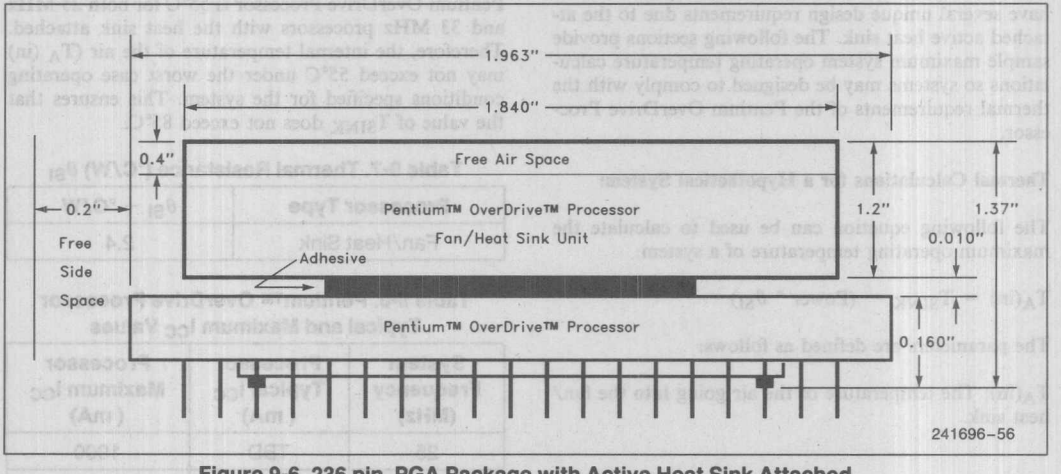


Figure 9-6. 236 pin, PGA Package with Active Heat Sink Attached

Table 9-6. Pentium™ OverDrive Processor, 236 Pin PGA Package Dimensions with Active Heat Sink Attached

Component	Length and Width (inches)		Height (inches)	
	Minimum	Maximum	Minimum	Maximum
PGA Package	1.950	1.975	0.140	0.180
Adhesive	N/A	N/A	0.008	0.012
Heat Sink Unit	1.830	1.850	N/A	N/A
Heat Sink	N/A	N/A	0.790	0.810
Req'd Free Space	N/A	N/A	0.400	0.400
External Total	**1.950**	**1.975**	**1.338**	**1.402**
Space from Package	0.200	0.200	N/A	N/A

Since the Pentium OverDrive Processor dissipates more power than the Intel486 CPU family members, it requires a larger cooling capacity. To facilitate the task of cooling the Pentium OverDrive Processor, Intel will ship the product with a fan/heat sink. No external connections (i.e., power) will be required for the fan/heat sink. All the needed connections will be made through the pins of the processor. The amount of extra power needed for the fan is accounted for in the I_{CC} numbers of the processor (see Section 9.2.3). To ensure adequate air circulation, the additional clearance specified in Table 9-6 must be provided.

9.2.3 THERMAL DESIGN CONSIDERATIONS

The Pentium OverDrive Processor and system chassis have several unique design requirements due to the attached active heat sink. The following sections provide sample maximum system operating temperature calculations so systems may be designed to comply with the thermal requirements of the Pentium OverDrive Processor.

Thermal Calculations for a Hypothetical System:

The following equation can be used to calculate the maximum operating temperature of a system:

$$T_A(in) = T_{SINK} - (Power * \theta_{SI})$$

The parameters are defined as follows:

$T_A(in)$: The temperature of the air going into the fan/heat sink.

T_{SINK}: Temperature of heat sink base, as measured in the center.

Power: Dissipation in Watts = $V_{CC} * I_{CC}$

θ_{SI}: Heat Sink to Internal Temperature [$T_A(in)$] Thermal Resistance

$T_A(out)$: The temperature of the air outside the system.

Since the Pentium OverDrive Processor uses an active heat sink, θ_{SI} (as shown in Table 9-7) is relatively constant, regardless of the airflow provided to the processor. Table 9-8 details the maximum current requirements of the Pentium OverDrive Processor. The maximum ambient temperature specification for the Pentium OverDrive Processor is 55°C for both 25 MHz and 33 MHz processors with the heat sink attached. Therefore, the internal temperature of the air (T_A (in) may not exceed 55°C under the worst case operating conditions specified for the system. This ensures that the value of T_{SINK} does not exceed 85°C.

Table 9-7. Thermal Resistance (°C/W) θ_{SI}

Processor Type	θ_{SI} – °C/W
Fan/Heat Sink	2.4

Table 9-8. Pentium™ OverDrive Processor Typical and Maximum I_{CC} Values

System Frequency (MHz)	Processor Typical I_{CC} (mA)	Processor Maximum I_{CC} (mA)
25	TBD	1900
33	TBD	2500

I_{CC} is dependent upon the V_{CC} level of the system, processor bus loading, software code sequences, and silicon process variations.

Maximum TA(in) is specified and can be verified by using the equation and parameters provided.

$T_A(in) = T_{SINK} - (Power * \theta_{SI})$

$T_A(in) = 85°C - ((2.5A * 5V) * 2.4\ °C/W)$

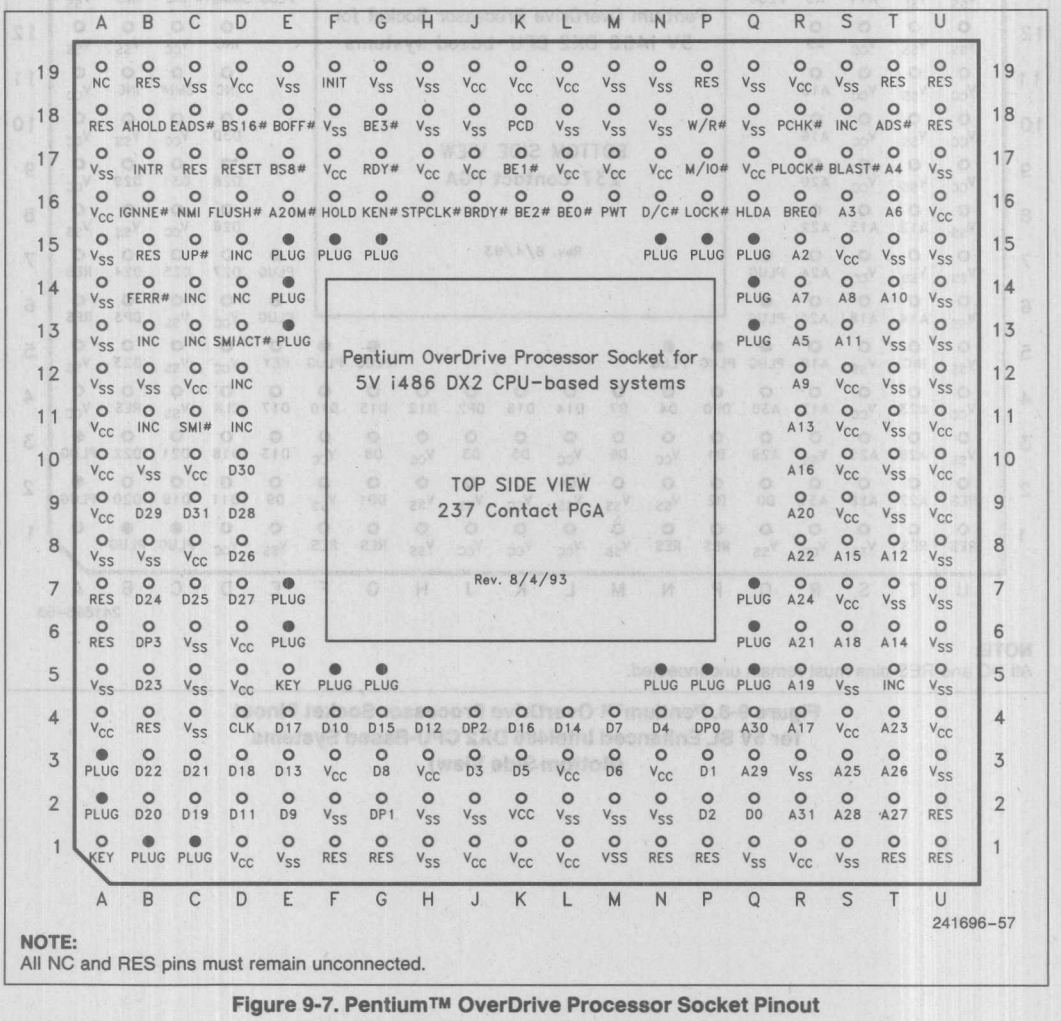
$T_A(in) = 85°C - (12.5\ Watts * 2.4\ °C/Watt)$

$T_A(in) = 85°C - 30°C$

$T_A(in) = 55°C$

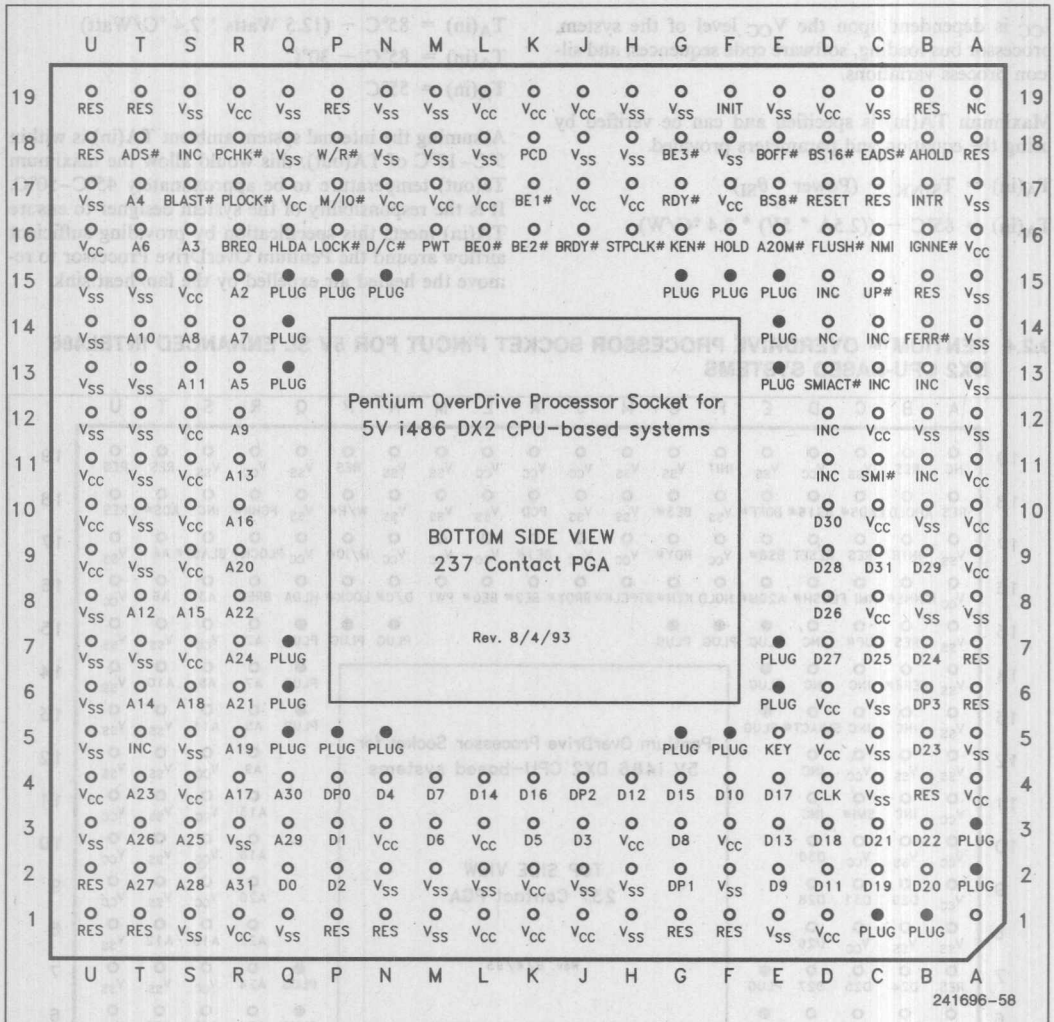
Assuming the internal system ambient TA(in) is within 5°C–10°C of TA(out), this would allow the maximum Ta(out) temperature to be approximately 45°C–50°C. It is the responsibility of the system designer to ensure TA(in) meets this specification by providing sufficient airflow around the Pentium OverDrive Processor to remove the heated air expelled by the fan/heat sink.

9.2.4 PENTIUM™ OVERDRIVE PROCESSOR SOCKET PINOUT FOR 5V SL ENHANCED INTEL486 DX2 CPU-BASED SYSTEMS



NOTE:
All NC and RES pins must remain unconnected.

Figure 9-7. Pentium™ OverDrive Processor Socket Pinout for 5V SL Enhanced Intel486 DX2 CPU-Based Systems (Top Side View)



NOTE:
All NC and RES pins must remain unconnected.

Figure 9-8. Pentium™ OverDrive Processor Socket Pinout for 5V SL Enhanced Intel486 DX2 CPU-Based Systems (Bottom Side View)

Table 9-9. 5V OverDrive Processor Socket Pin Cross Reference for Intel486 DX2 CPU-Based Systems

Address		Data		Control		Control		RES*	V_{CC}		V_{SS}	
A2	R15	D0	Q2	A20M#	E16	RDY#	G17	A6	A4	L1	A5	M18
A3	S16	D1	P3	ADS#	T18	RESET	D17	A7	A9	L3	A8	M19
A4	T17	D2	P2	AHOLD	B18	SMI#	C11	A18	A10	L17	A12	N2
A5	R13	D3	J3	BE0#	L16	SMIACT#	D13	A19	A11	L19	A13	N18
A6	T16	D4	N4	BE1#	K17	STPCLK#	H16	B4	A16	M17	A14	N19
A7	R14	D5	K3	BE2#	K16	UP#	C15	B15	C8	N3	A15	Q1
A8	S14	D6	M3	BE3#	G18	W/R#	P18	B19	C10	N17	A17	Q18
A9	R12	D7	M4	BLAST#	S17			C17	C12	Q17	B8	Q19
A10	T14	D8	G3	BOFF#	E18			F1	D1	R1	B10	R3
A11	S13	D9	E2	BRDY#	J16			G1	D5	R19	B12	S1
A12	T8	D10	F4	BREQ	R16			N1	D6	S4	C4	S5
A13	R11	D11	D2	BS8#	E17			P1	D19	S7	C5	S19
A14	T6	D12	H4	BS16#	D18			P19	F3	S9	C6	T7
A15	S8	D13	E3	CLK	D4			T1	F17	S10	C19	T9
A16	R10	D14	L4	D/C#	N16			T19	H3	S11	E1	T10
A17	R4	D15	G4	DP0	P4	**Position**		U1	H17	S12	E19	T11
A18	S6	D16	K4	DP1	G2	KEY	E5	U2	J1	S15	F2	T12
A19	R5	D17	E4	DP2	J4	KEY	A1	U18	J17	U4	F18	T13
A20	R9	D18	D3	DP3	B6	PLUG	A2	U19	J19	U9	G19	T15
A21	R6	D19	C2	EADS#	C18	PLUG	A3	**N/C***	K1	U10	H1	U3
A22	R8	D20	B2	FERR#	B14	PLUG	B1	A19	K2	U11	H2	U5
A23	T4	D21	C3	FLUSH#	D16	PLUG	C1	D14	K19	U16	H18	U6
A24	R7	D22	B3	HLDA	Q16	PLUG	E6	**INC**			H19	U7
A25	S3	D23	B5	HOLD	F16	PLUG	E14	B11			J2	U8
A26	T3	D24	B7	IGNNE#	B16	PLUG	E15	B13			J18	U12
A27	T2	D25	C7	INIT	F19	PLUG	F5	C13			L2	U13
A28	S2	D26	D8	INTR	B17	PLUG	F15	C14			L18	U14
A29	Q3	D27	D7	KEN#	G16	PLUG	P5	D11			M1	U15
A30	Q4	D28	D9	LOCK#	P16	PLUG	P15	D12			M2	U17
A31	R2	D29	B9	M/IO#	P17	PLUG	Q5	D15				
		D30	D10	NMI	C16	PLUG	Q6	S18				
		D31	C9	PCD	K18	PLUG	Q14	T5				
				PCHK#	R18	PLUG	Q15					
				PLOCK#	R17							
				PWT	M16							

* All RES pins are reserved for later use by Intel. To ensure proper operation of the microprocessor, all RES and N/C pins should be left unconnected. Please contact Intel for design information.

9.2.5 D.C./A.C. CHARACTERISTICS

The 5V Pentium OverDrive Processor is compatible with the A.C. specifications for the 5V SL Enhanced Intel486 DX2 CPU (maximum I_{CC} current values provided in Section 9.2.3).

10.0 REVISION HISTORY

Revision -002 of the SL Enhanced Intel486 Microprocessor Data Sheet Addendum contains many updates and improvements to the original version. The sections significantly revised since version -001 are:

Table 1-1	Added 3.3V 50 MHz version of the Intel486 DX2 CPU.
Table 2-3	Added note to explain that pins A3, A14, B14 and B16 are Boundary Scan pins for the 50 MHz version of the standard Intel486 DX CPU.
Table 2-4	Corrected PGA control pins, F17 and A10.
Table 2-5	Added new table to show pinout differences between the SL Enhanced Intel486 SX CPU and the SL Enhanced Intel486 DX and DX2 CPUs for pins 66 and 72.
Table 2-7, Sections 3.2 and 7.3.2	Added clarification for the STPCLK# signal with respect to the Stop Grant bus cycle.
Figure 3-1	Modified figure to show that STPCLK# must be deasserted for at least 5 clocks after RDY# becomes active.
Section 3.5.6	Added statement about Auto Idle Power Down.
Section 4.3.1	Added clarification about the SMI# specification.
Section 4.3.3.1 and Table 4-1	Added information about SMRAM state save map.
Sections 6.0 to 6.4	Changed T_{case} specification guarantee of 90°C at zero airflow to 85°C.
Table 6-1	Added max. D.C. spec. of 6 pF for C_{CLK}.
Table 6-2	Changed 3.3V Intel486 SX-33 CPU typical and maximum I_{CC} specifications.
Table 6-4	Added I_{CC} values for 3.3V Intel486 DX2 CPU.
Figure 6-1	Added 3.3V Intel486 DX2-50 CPU to graph.
Table 6-11	Added minimum CLK frequency specifications for 50 and 66 MHz versions of 5V Intel486 DX2 CPU.
Tables 6-15a and 6-15b	Changed θ_{JA} and θ_{JC} specifications for Intel486 DX and DX2 CPUs.
Tables 6-20, 6-21 and 6-22	Recalculated $T_{ambient}$ values for Intel486 SX, DX and DX2 CPUs based on T_{case} of 85°C, new θ_{JA}, θ_{JC} and I_{CC} specifications.
Section 6.4	Added capacitive derating information section.
Figures 9-4 and 9-5	Corrected pinouts of the OverDrive processor for 5V Intel486 SX and DX CPUs.
Section 9.2	Modified 5V OverDrive Processor Socket section to reflect new Pentium™ OverDrive Processor Socket specifications for SL Enhanced Intel486 DX2 CPU-based systems.
Section 10.0	Added revision history section to reflect changes since -001 version of publication.

APPENDIX A
SYSTEM DESIGN NOTES

A.1 SMM Environment Initialization

When the SL Enhanced Intel486 CPUs are operating in Real Mode, the physical address at which instructions and data are fetched is determined by the segment register and an offset (i.e., CS and IP for instructions). When a new value is loaded into a segment register, the new value is shifted to the left by four bits and stored in a segment base register that corresponds to that particular segment (CSBASE, DSBASE, ESBASE, etc.). It is the value stored in the segment base register that is actually used to generate a physical address. For example, the linear address to be used for fetching instructions is determined by adding the value contained in the CS segment base register with the value in the IP register.

When the CPU is in Protected Mode, the segment registers are used as selectors to a descriptor table. Each descriptor in a descriptor table contains information about the segment in use, including the segments BASE address (i.e., CSBASE), the limit (or size of the segment), as well as protection level, privileges, operand sizes, and the segment type. In Protected Mode, the linear address is determined by adding the base portion of the descriptor to the appropriate offset.

When in System Management Mode, the CPU operates in a pseudo-Real Mode, with address calculation performed in the Real Mode manner. However, the CPU adds the value in the segment base register with the value in the EIP register, rather than the IP register, so there are no limits as to the segment size. The physical address of an instruction is obtained by adding the value in CSBASE to the value in EIP.

When entering SMM, it may be necessary to initialize the segment registers to point to SMRAM (see section 4.4.2. for their value on SMM entry). If SMBASE has not been relocated, then the necessary segment registers can be initialized to point to SMRAM by using the value in the CS register, 3000H, which points to the SMRAM address space.

When an SMI# occurs after SMBASE has been modified, CSBASE is loaded with the new value of SMBASE. **However, the CS selector register still contains the value 3000H, not the value corresponding to the new SMBASE.**

In order to initialize the segment registers to point to the new SMRAM area, we need to read the value of SMBASE from the SMM state save in memory. Since the data segment registers are initialized to 0, we cannot use them to access the SMM state save area. However, we can perform a read relative to the CS register by using a CS override prefix to a normal memory read. Although CS still contains 3000H, CSBASE contains the value of SMBASE, and CSBASE is used for the address generation.

Once the value of SMBASE is obtained, it must be shifted to the right by four bits to get the appropriate value to be placed in the segment registers. The CS register itself can be initialized by executing a far jump instruction to an address within SMBASE, which causes CS to be reloaded with a value corresponding to SMBASE.

Figure A-1 describes one method of initializing the segment registers when SMBASE has been relocated. This method works if SMBASE is less than 1 Megabyte.

```
;read the value of SMBASE from the state save area

      mov         si,FEF8H              ;SMBASE slot in SMM state save area
      mov         eax,cs:[si]           ;copy SMBASE from SMBASE:FEF8H to eax

;scale the SMBASE value to a 16-bit quantity

      mov         cl,4
      ror         eax,cl                ;scaled value of SMBASE now in ax

;to load cs, we need to execute a far jump to an address that has been stored
;at memory location PTR_ADDR

;store the SMBASE value and an offset to a memory location that can be used as
;an indirect jump address

      mov         di,PTR_ADDR           ;PTR_ADDR is the location used to
                                        ;store the jump address
      mov         bx,OFFSET             ;OFFSET is the address where
                                        ;execution continues after the
                                        ;far jump

      mov         cs:[di],bx            ;store the offset for the far jump
      inc         di
      inc         di
      mov         cs:[di],ax            ;store the segment address for the
                                        ;far jump, which is SMBASE
      mov         bx,PTR_ADDR           ;bx now contains the address of the
                                        ;location holding the jump address

;initialize DS and ES with the correct address of SMBASE

      mov         ds,ax
      mov         es,ax

;execute a far jump instruction to load the CS register

      jmp         far [bx]              ;jump to address stored at memory
                                        ;location pointed to by bx

;CS now contains the correct value of SMBASE, and execution continues from the
;address SMBASE:OFFSET
```

Figure A-1. Initialization of Segment Registers Within SMM

A.2 Accessing SMRAM

A.2.1 LOADING SMRAM WITH AN INITIAL SMI HANDLER

Under normal conditions, the SMRAM address space should only be accessible by the CPU while it is in SMM mode. However, some provision must be made for providing the initial SMM interrupt handler routine.

Since System Management Mode must be transparent to all operating systems, the initial SMM handler must be loaded by the system BIOS. At some time during the power on sequence, the system BIOS will need to move the SMM handler routine from the BIOS ROM to the SMRAM. The system designer must provide a hardware mechanism that allows access to SMRAM while SMIACT# from the CPU is inactive. One method would be to provide an I/O port in the memory controller that forces memory cycles at a given address to be relocated to the SMRAM. Once the initial SMM handler has been loaded to SMRAM, the I/O port would be disabled to protect against accidental accesses to SMRAM.

The system BIOS must provide an SMM handler at the address 38000H. If the system designer has chosen to take advantage of the SMRAM relocation feature of the CPU, this handler must change the SMBASE register in the SMM state save. Next, the BIOS must move the full featured SMM handler to the new address. An SMI# must be generated in order to change the SMBASE register before the BIOS passes control to the operating system.

A.2.2 SMRAM HIDDEN FROM DMA AND BUS MASTERS

In a system that allows DMA or other devices to take control of the system bus, care must be taken to ensure that only the master CPU can access SMRAM. If an external bus master requests use of the system bus (by asserting HOLD or BOFF#) while the CPU is executing an SMM handler routine, the CPU would respond by passing control of the bus to the requesting device. The system memory controller must redirect any memory accesses that are not generated by the CPU to normal system memory as if SMIACT# was inactive.

DMA accesses to the SMRAM area must be redirected to the correct address space when the initialization routine is loading SMRAM, as well as when the CPU is in SMM.

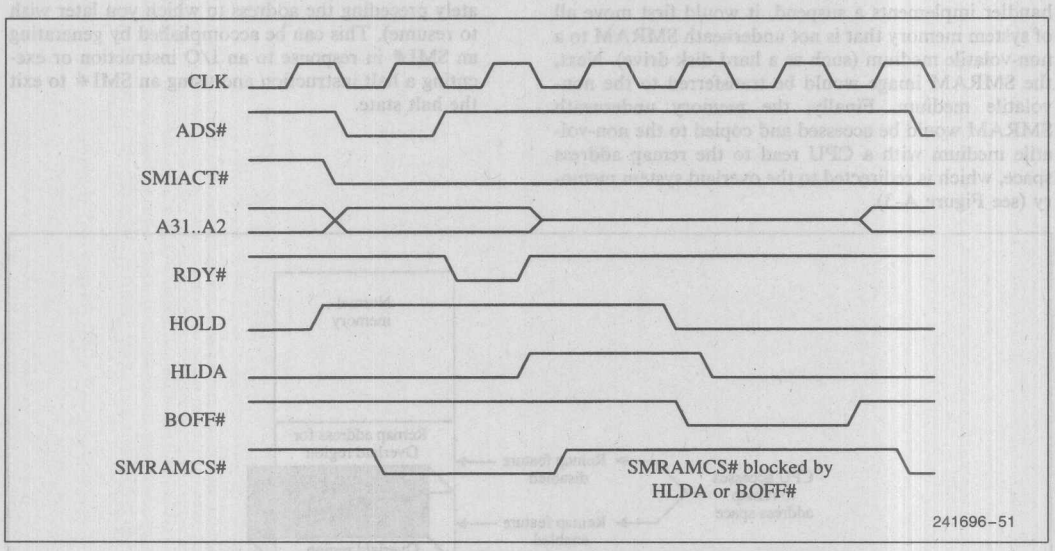


Figure A-2. Blocking Other Bus Masters from Accessing SMRAM

It is not recommended to block bus control requests when in SMM, since the increased bus access latency could cause compatibility issues with some software or expansion hardware.

A.2.3 ACCESSING SYSTEM MEMORY FROM WITHIN SMM

In order to enter a suspend state where power is removed from some or all of system memory, it is necessary for the CPU to have access to the entire system address space from within SMM. Access to system memory from within SMM requires that the memory controller decode both SMIACT# and the CPU address to determine accesses to SMRAM. Only those memory addresses that are defined as being SMRAM space would be directed to SMRAM. If SMRAM is located at an address that overlays normal system memory address space (see section 4.6.2.), the CPU must have a method of accessing both SMRAM (for code reads) and system memory simultaneously.

Ideally, a method of accessing system memory that is mapped underneath SMRAM would be provided by the system memory controller. The memory controller would provide a register that allows system memory at a given address to be remapped to a different address, which is not overlaid by SMRAM. When the SMM handler implements a suspend, it would first move all of system memory that is not underneath SMRAM to a non-volatile medium (such as a hard disk drive). Next, the SMRAM image would be transferred to the non-volatile medium. Finally, the memory underneath SMRAM would be accessed and copied to the non-volatile medium with a CPU read to the remap address space, which is redirected to the overlaid system memory (see Figure A-3).

If the memory controller does not provide a method of accessing overlaid system memory, it is possible to implement a software procedure to accomplish the same goal. However, the software method is quite complex, and a hardware method is preferred. A description of the software method follows.

The ability to access the system memory that is located in the address space under SMRAM requires a method of resuming from SMM to a predetermined address space. This can be accomplished with the following procedure.

When resuming from SMM, the CPU continues execution at the address contained in the CS and EIP slots within the SMM state save. However, the resume address cannot be changed by simply modifying the CS and EIP slots, since the CPU will use the CS descriptor to determine the actual resume address. The descriptor registers are stored in reserved slots in the SMM state save, and they cannot be directly modified.

By replacing the suspend state save with a previously obtained image of a state save that returns to a known location, the SMM suspend handler can force a return to a given address:

1. During initial system power up, execute an SMI# from a predetermined address (the address immediately preceding the address to which you later wish to resume). This can be accomplished by generating an SMI# in response to an I/O instruction or executing a halt instruction and using an SMI# to exit the halt state.

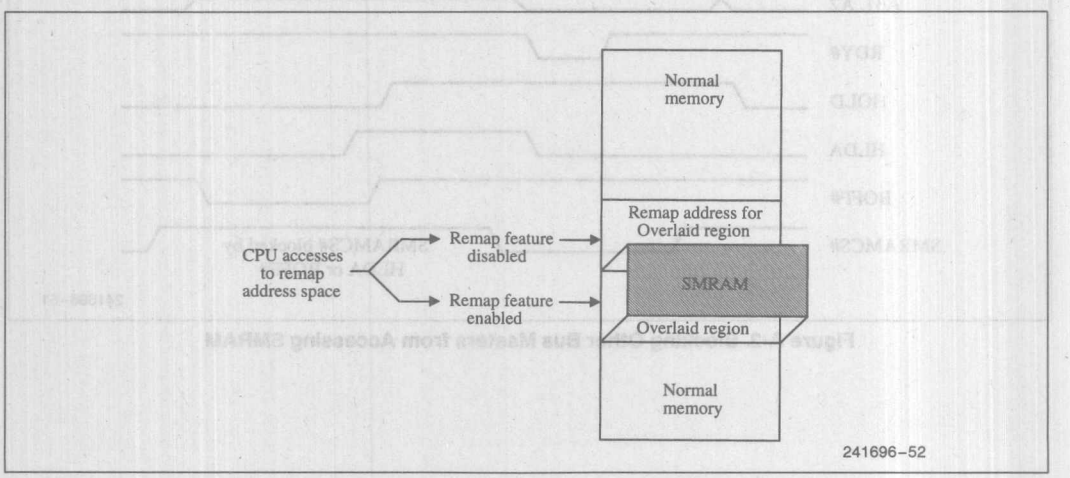


Figure A-3. Remapping Memory that is Overlaid by SMRAM

2. Save the state save from this SMM to a safe location (SMRAM).
3. When the system needs to resume to a given address from some other SMI#, the stored state save can be substituted for the state save generated from that particular SMM.

Now that we have the ability to resume from SMM to a predetermined address, we can access the entire system memory space from within SMM before executing a suspend:

1. During a suspend SMM, save all system memory except that which is located underneath SMRAM to a specified (and reserved) section of the hard disk. The ability to access system memory requires the memory controller to decode both SMIACT# and the CPU address, and direct a limited section (maybe 64 or 128K) of the CPU address space to SMRAM. All other CPU memory accesses should go to normal system memory.
2. Save the contents of the SMM state save to the hard disk.
3. Modify the SMM state save so that the RSM instruction will return to a predefined address, which is not in the application that was interrupted. The code at this address must contain the remainder of the suspend SMM handler. The predefined address can be anywhere in the CPU address space, since the contents of system memory have already been saved to disk.
4. Execute a RSM instruction, which exits SMM and returns control to a predetermined address (which must contain the rest of the SMM suspend handler).
5. Save the rest of system memory (that which is located underneath SMRAM) to the hard disk. This address space can now be accessed with normal move instructions, since we are no longer in SMM.
6. Save a flag (in CMOS memory) indicating that the next reset should cause a resume from suspend.
7. Powerdown the memory (and possibly the CPU).
8. When power is restored, the CPU is reset and begins execution of the POST in BIOS. Early in the POST, the system should check the status of the suspend flag.
9. Load a preliminary SMM handler to location 38000H and generate an SMI#. The SMM handler should read the SMBASE slot from the SMM state save that was stored to hard disk. SMBASE is then modified to point to the final SMRAM location and the system resumes from SMM back to the system BIOS.
10. Restore the contents of system memory located underneath SMRAM from the hard disk.
11. Generate a second SMI#, which executes an SMM handler at the original value of SMBASE (before the suspend SMM). The SMM handler restores the contents of the rest of system memory from the hard disk, and then restores the original SMM state save to the SMM state save area in SMRAM, discarding the most recent SMM state save.
12. Execute an RSM instruction, which returns execution to the application that was interrupted by the suspend request.

A.3 Interrupts and Exceptions During SMM Handler Routines

To ensure transparency to existing system software, the SMM handler should not depend on interrupt or exception handlers provided by the operating system. However, in some cases it may be necessary to service interrupts or exceptions while in System Management Mode. In these cases, SMM compliant interrupt and exception handlers, as well as an SMM compliant interrupt vector table, should be provided.

A.3.1 SMM COMPLIANT VECTOR TABLES

An SMI# interrupt request can be generated while code is running under any of the other three CPU operating modes (Real, Virtual-86, or Protected). When entering the SMM handler, the CPU enters a pseudo-real mode, and the beginning of the interrupt vector table must be located at the address 00000000H. Before allowing any interrupts or exceptions to occur, the SMM handler routine must provide a valid interrupt vector table. Any code that is executed before setting up an SMM compliant interrupt vector table must be written carefully to ensure that no exceptions are generated.

The system memory controller could relocate accesses to the SMM interrupt vector table to a location within SMRAM. In this case, when SMIACT# is active, all accesses to the lowest 1 Kbyte of the CPU address space would be redirected to SMRAM, which would contain an SMM compliant vector table that has already been initialized.

If the system memory controller does not redirect interrupt vector table reads to an address within SMRAM, there are three steps required to provide an SMM compliant interrupt vector table:

1. Save the contents of memory at address 00000000H to SMRAM
2. Provide vectors for any possible interrupts or execptions at the apropriate location in the vector table
3. Restore the original memory contents from SMRAM before exiting the SMM handler routine

A.3.2 INTERRUPTS AND SUBROUTINES WITH SMRAM RELOCATION

There is an additional issue that must be considered if interrupts or exceptions are to be executed within SMM and SMRAM has been relocated. Interrupt or subroutine calls from within SMM operate in a manner similar to Real Mode. When a subroutine is called or an interrupt is recognized, the 16-bit CS and IP registers are pushed onto the stack to provide a return address.

When SMRAM is relocated to an address space above 1M and an interrupt or subroutine call occurs, only 16 bits of the EIP register are pushed onto the stack. When returning from the subroutine or interrupt, the CPU will vector to a location where the upper 16 bits of EIP are zero. This can be avoided for subroutines by using an address size override before calling the subroutine. However, the issue remains for interrupts.

A.4 Floating Point Operation and SMM

A.4.1 THE NEED TO SAVE THE FPU ENVIRONMENT

When the CPU enters System Management Mode, the context information for the interrupted application is automatically saved to a specific state save address. When the SMM handler returns control to the interrupted application by executing the RSM instruction, the context information from the interrupted application is restored to the CPU by reading from the state save location. This mechanism allows the SMM handler routine to modify most of the CPU registers without the need to explicitly save them to memory. However, the registers in the CPU's Floating Point Unit (FPU) are not automatically saved when the CPU enters SMM. If the SMM handler needs to modify any of the registers in the FPU, or if the register data will be lost due to entering a power down state, the SMM handler must first explicitly save the FPU state as it existed in the interrupted application.

There are two instances in which an SMM handler routine must be aware of the Floating Point Unit (FPU):

1. When removing power from the CPU / FPU for the purpose of executing a suspend sequence.
2. When the SMM handler uses FPU instructions.

In both of these cases, the SMM handler must save the state of the FPU as it was left by the interrupted application.

The information stored by the FPU state save instructions (FSAVE, FNSAVE, FSTENV, and FNSTENV) is dependent on the operating mode of the CPU. The FPU state save instructions store the FPU state information in one of four formats: 16-bit Real Mode, 32-bit Real Mode, 16-bit Protected Mode, or 32-bit Protected Mode, depending on the CPU operating mode. The content of the information saved also varies slightly, depending on the CPU operating mode in which the save instruction was executed. For example, the 32-bit Protected Mode FNSAVE instruction saves the address of the last executed FPU instruction and its operands in the form of a segment selector and a 32-bit offset. In contrast, the 16-bit Real Mode FNSAVE instruction saves the address information in the form of a 20 bit physical address. Since the format with which the FPU state restore instructions (FRSTOR and FLDENV) recall the information is also dependent on the operating mode of the CPU, the save and restore instructions must be executed from the same CPU operating mode.

A.4.2 SAVING THE STATE OF THE FLOATING POINT UNIT

When an SMM handler routine needs to save the state of the Floating Point Unit, it must save all FPU state information necessary for the interrupted application to continue processing. This state information includes the contents of the Floating Point Unit stack, which requires use of the FNSAVE or FSAVE instruction (FSTENV does not save the contents of the FPU stack). If the last executed non-control Floating Point instruction caused an error (such as a divide by 0), the saved information must also include the address of the failing instruction and the addresses of any operands for that instruction. Without these addresses, it would be impossible for the FPU exception handler of the interrupted application to correct the error and restart the instruction.

The FNSAVE and FSAVE instructions differ in that FNSAVE does not wait for the FPU to check for an existing error condition before storing the FPU environment information. If there is an unmasked FPU exception condition pending, execution of the FSAVE instruction will force the CPU to wait until the error condition is cleared by the software exception handler. Because the CPU is in System Management Mode, the appropriate exception handler will not be available, and the FPU error would not be corrected in the manner expected by the interrupted application program. For this reason, the FNSAVE instruction should be used when saving the environment of the FPU within SMM.

Since the SMM handler does not know the CPU mode in which the interrupted application was executing (16 or 32 bit, Real or Protected), the SMM handler must execute the FNSAVE instruction in a mode in which

all FPU state information is stored. The 32-bit Protected Mode format of the FNSAVE instruction is a super set of all other formats of the FNSAVE instruction. Therefore, executing the 32-bit Protected Mode FNSAVE instruction ensures that all FPU state information will be saved.

Executing the FNSAVE instruction in 32-bit Protected Mode requires that the CPU be temporarily placed in Protected Mode. Rather than perform all of the setup details and overhead necessary to place the CPU into Protected Mode, including the initialization of all descriptors and descriptor tables, it is possible to temporarily place the CPU into Protected Mode for the purpose of executing only a few carefully written instructions. This can be accomplished by setting the PE bit in the CR0 register, and then executing a short jump to clear the instruction pipelines.

It is important to note that any instruction that modifies a segment register will cause the CPU to attempt to load a new descriptor from the descriptor table. (The occurrence of an interrupt or an exception would cause the CPU to load a new descriptor, so interrupts must be disabled during this sequence.) Since neither the descriptors nor the descriptor table have been initialized, this would cause the system to crash. Therefore, all segment registers that are to be used in the FPU state save instructions must be initialized before entering Protected Mode.

Figure A-4 gives an example of the code that can be used to place the CPU in Protected Mode and save the FPU state.

Note that the no wait form (FNSAVE) of the save instruction must be used. In the event that the previous FPU instruction caused a floating point error, we do not want to wait for this error to be serviced before executing the save instruction. Additionally, if the FSAVE instruction were used, the operand size override prefix would be incorrectly applied to the implicit WAIT instruction which precedes FSAVE, rather than to the save instruction itself.

Before exiting the SMM handler and returning to the interrupted application, the register contents of the Floating Point Unit must be returned to their previous values. This can be accomplished by executing the 32-bit Protected Mode format of the FRSTOR instruction. Figure A-5 gives an example code segment that can be used to restore the FPU to the state in which it was interrupted by the SMI request.


```
;first initialize the registers used to store the state save information

        mov             dx,SEGMENT          ;SEGMENT is the segment to be used by
                                            ;the save instruction,
        mov             ds,dx               ; normally it should point to SMRAM
        mov             si,OFFSET           ;OFFSET is the offset used in the save
                                            ;instruction

;set the PE bit in CR0

        mov             eax,cr0             ;read the old value of CR0
        or              eax,00000001H       ;set the PE bit
        mov             cr0, eax

;enter protected mode by executing a short jump to clear the prefetch queue

        jmp             protect
protect:

;we can now save the state of the FPU in the protected mode format

        db              66H                 ;use an operand size override prefix
                                            ;to set 32-bit format
        fnsave          [si]                ;FPU state saved to SEGMENT:OFFSET

;now return to real mode to continue with the SMM handler (no jump is
;required)

        mov             eax,cr0             ;clear the PE bit in CR0
        and             eax,0FFFFFFFEH
        mov             cr0,eax
```

Figure A-4. Saving the FPU State in 32-bit Protected Mode

```
;first initialize the registers used to recall the state save information
        mov             dx,SEGMENT          ;SEGMENT is the segment to be used by
                                            ;the restore instruction,
        mov             ds,dx               ; normally it should point to SMRAM
        mov             si,OFFSET           ;OFFSET is the offset used in the
                                            ;restore instruction

;set the PE bit in CR0
        mov             eax,cr0             ;read the old value of CR0
        or              eax,00000001H       ;set the PE bit
        mov             cr0, eax

;enter protected mode by executing a short jump to clear the prefetch queue
        jmp             protect
protect:

;we can now recall the state of the FPU from the previous FNSAVE instruction
;(in the protected mode format)
        db              66H                 ;use an operand size override prefix
                                            ;to set 32-bit format
        fnrstor         [si]                ;FPU state restored from
                                            ;SEGMENT:OFFSET

;now return to real mode to continue with the SMM handler (no jump is
;required)
        mov             eax,cr0             ;clear the PE bit in CR0
        and             eax,FFFFFFFEH
        mov             cr0,eax
```

Figure A-5. Restoring the FPU State from a 32-bit Protected Mode Save

Note that the no wait form (FNRSTOR) of the restore instruction must be used. If the FRSTOR instruction were used, the operand size override prefix would be incorrectly applied to the implicit WAIT instruction which precedes FRSTOR, rather than to the save instruction itself.

A.5 Support for Power Managed Peripherals

A.5.1 SHADOW REGISTERS

Before power is removed from any device, the state of that device must be saved in a protected memory space so that the device can be reinitialized to its previous state. If a peripheral contains a write only register, the value in that register can be recovered by providing shadow registers that are both readable and writeable. These shadow registers would be updated every time the peripheral registers are written, but they have no function other than tracking the data written to a particular register.

In addition to the write only registers in a system, there are several other registers that must be shadowed. Any device that requires registers to be programmed in a particular sequence must also have its registers shadowed. Examples in a typical personal computer include the programmable interrupt controller, the DMA controller, and the programmable timer/counter.

It is also possible to perform shadowing of some write only registers using SMM. Any time a write cycle is generated to a write only register, the system can generate an SMI#. The SMM handler can use the CPU state information saved in the SMM state save to save the data from the interrupted I/O cycle to a predetermined location in the SMRAM space.

The information contained in the SMM state save can be used (with the knowledge that the SMI# was in response to an I/O write instruction) to determine both the address and the data of the interrupted write instruction. The SMM handler can examine the OPCODEs of previous instructions by decrementing the IP (or EIP) register. Once the correct OPCODE is determined, it can be used with the values in the EAX and DX slots of the SMM state save to update the information in the memory used to shadow the I/O register. I/O write instructions occur in one of three forms: 1) a write to an address that is specified in the OPCODE; 2) a write to an address contained in the DX register; or 3) a string write to an address contained in the DX register.

The I/O write instructions have the following OPCODEs:

Table A-1. I/O Write Instruction OPCODEs

Instruction	OPCODE	Notes
OUT x,al	E6 x	x is the address of the I/O port
OUT x,ax	E7 x	x is the address of the I/O port
OUT x,eax	E7 x	x is the address of the I/O port
OUT dx,al	EE	
OUT dx,ax	EF	
OUT dx,eax	EF	
OUTSB	6E	
OUTSW	6F	
OUTSD	6F	

The SMM handler must know whether a particular I/O port is 16 or 32 bits in order to distinguish between 16 and 32 bit I/O write cycles.

The SMM handler can decrement the value of IP contained in the state save, and then examine the memory contents at that address. If the SMM handler knows that the last instruction was an I/O write instruction, and writes to I/O addresses 6EH, 6FH, 0EEH, and 0EFH will not cause an SMI#, it can use the SMM state save data for EAX and EDX to reconstruct the last instruction.

A.5.2 HANDLING INTERRUPTED I/O WRITE SEQUENCES

In a typical personal computer, there are several hardware devices that require the control registers for that device to be programmed in a particular order. For example, the interrupt controller, the DMA controller, the programmable timer/counter, the keyboard controller, and the real time clock all require a series of accesses to properly initialize the registers in that particular device. Some of these devices may require successive accesses to registers located at different addresses, while others may require several control registers to be programmed through write cycles to the same address.

If an SMI request interrupts an application that is in the process of initializing the registers in one of these devices, special care must be taken to ensure that the peripheral is returned to its original state when control is returned to the interrupted program. For some SMM handler events, it may be necessary to power down the device or change the state of a register within the device. In these cases, the SMM handler must return control to the interrupted application in such a way that the application can continue with the correct sequential access in the interrupted sequence.

To accomplish this, the SMM handler must restore the original values of all registers in the device, and restart the interrupted sequence so that the application may continue where it left off. This requires system hardware to shadow all registers that need to be accessed in the sequence, keep an index indicating which position in the sequence the register occupies, and keep a pointer so that SMM software knows to which register the last access was directed. This pointer would indicate the last register of each sequence that was programmed in the particular peripheral.

For example, programming the master interrupt controller requires a write to I/O port 20H (ICW1) followed by four write cycles to I/O port 21H (ICW2, ICW3, ICW4, and OCW1). If this sequence is interrupted by an SMI request, and the resulting SMM handler either modifies or powers down the interrupt controller, the SMM handler must return control to the interrupted application such that the following access to the interrupt controller would access the correct register in the sequence. System hardware must save the contents of each of the registers, as well as a pointer indicating which register was last written.

Before returning control to the interrupted application, the SMM handler must initialize ICW1-ICW4 and OCW1 to their previous values. It would then re-write the appropriate registers so that the first access by the application program would be to the location in the sequence following the last location it programmed before it was interrupted by the SMI request.

A similar procedure must be followed for each of the peripherals that require control registers to be initialized in a particular order.

3.3V Intel486™ SX MICROPROCESSOR

- **Lower Supply Voltage Required**
  - — Operates from 3.0V to 3.6V
  - — 50% Lower Power than 5V CPU
  - — Lower Power Consumption Produces Less Heat
  - — Longer Battery Life for Portable Applications
- **Binary Compatible with Large Software Base**
  - — MS-DOS*, OS/2**, Windows*
  - — UNIX*** System V/386
  - — iRMX®, iRMK Kernels
- **High Integration Enables On-Chip**
  - — 8 Kbyte Code and Data Cache
  - — Paged, Virtual Memory Management
- **Easy To Use**
  - — Built in Self Test
  - — Intel Software Support
  - — Extensive Third Party Software Support
- **196-Lead PQFP Package**
- **High Performance Design**
  - — Intel486 One Clock Instruction Core
  - — 25 MHz, 20 MHz, and 16 MHz Clock Frequencies
  - — 80 Mbytes/sec Burst Bus at 25 MHz
  - — CHMOS V Process Technology
  - — Dynamic Bus Sizing for 8-, 16-, 32-Bit Busses
- **Complete 32-Bit Architecture**
  - — Address and Data Busses
  - — Registers
  - — 8-, 16-, 32-Bit Data Types
- **IEEE 1149.1 Boundary Scan Compatibility**
  - — For Surface Mount Production Testing

The 3.3V Intel486 SX microprocessor provides a low-cost entry point to powerful Intel486 microprocessor based portable computing. The Intel486 SX microprocessor is a binary compatible derivative of the Intel486 DX microprocessor, giving it access to the $40 billion installed software base of over 50,000 MS-DOS, Windows, OS/2, and UNIX system V/386 applications. The Intel486 SX microprocessor has the same integrated RISC integer core, 8 Kbyte cache memory, and memory management unit as the Intel486 DX microprocessor.

The high-performance RISC integer core of the Intel486 SX microprocessor executes frequently-used instructions in one clock cycle. An 8 Kbyte unified code and data cache allow this performance level to be sustained. An 80 Mbyte/sec burst bus at 25 MHz ensures high system throughput even with inexpensive DRAMs. The 25 MHz Intel486 SX microprocessor has a Norton SI V5.0 rating of 54.1, a Dhrystone MIPS rating of 20.1 and a 13.3 SPEC Integer rating. It provides up to 70% greater performance than a 33 MHz Intel386™ DX microprocessor with an external cache (depending on the application) at a lower system cost.

This documentation lists the A.C. and D.C. specifications of the 3.3V Intel486 SX microprocessor and is a supplement to the Intel486™ SX Microprocessor/Intel487™ SX Math Coprocessor Data Book (Order Number 240950-002).

---

i386, Intel386, i387, Intel387, i486, Intel486, i487, Intel487 are trademarks of Intel Corp.
*MS-DOS and Windows are trademarks of Microsoft Corp.
**OS/2 is a trademark of IBM.
***UNIX is a registered trademark of UNIX Systems Labs

June 1992
Order Number: 290443-001

Maximum Ratings

The ratings listed in this section are stress ratings only, and functional operation at the maximums is not guaranteed. Functional operating conditions are given in the D.C. Specifications and A.C. Specifications sections.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the Intel486 SX microprocessor contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

Absolute Maximum Ratings

Case Temperature under Bias	−65°C to +110°C
Storage Temperature	−65°C to +150°C
Voltage on Any Pin with Respect to Ground	−0.5V to V_{CC} + 0.5V
Supply Voltage with Respect to V_{SS}	−0.5V to +6.5V

D.C. SPECIFICATIONS

Functional operating range: V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C

2

3.3V Intel486™ SX Microprocessor D.C. Parametric Values

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input LOW Voltage	−0.3	+0.8	V	
V_{IH}	Input HIGH Voltage	2.0	V_{CC} + 0.3	V	
V_{OL}	Output LOW Voltage I_{OL} = 4/5 mA I_{OL} = 0.1 mA		 0.45 0.2	 V V	(Note 1)
V_{OH}	Output HIGH Voltage I_{OH} = −1/0.9 mA I_{OH} = −0.1 mA	 2.4 V_{CC} − 0.2		 V V	(Note 2)
I_{CC}	Supply Current CLK = 8 MHz CLK = 16 MHz CLK = 20 MHz CLK = 25 MHz		 150 260 290 340	 mA mA mA mA	(Note 3)
I_{LI}	Input Leakage Current		±15	µA	(Note 4)
I_{IH}	Input Leakage Current		200	µA	(Note 5)
I_{IL}	Input Leakage Current		−400	µA	(Note 6)
I_{LO}	Output Leakage Current		±15	µA	
C_{IN}	Input Capacitance		10	pF	F_C = 1 MHz(7)
C_{OUT}	Output or I/O Capacitance		10	pF	F_C = 1 MHz(7)
C_{CLK}	CLK Capacitance		6	pF	F_C = 1 MHz(7)

PRELIMINARY
SEE INTEL FOR DESIGN-IN
INFORMATION

NOTES:

1. This parameter is measured at:
   Address, Data, BEn 4.0 mA
   Definition, Control 5.0 mA
2. This parameter is measured at:
   Address, Data, BEn −1.0 mA
   Definition, Control −0.9 mA
3. Typical supply current (3.3V):
   ICC @ 8 MHz = 90 mA
   @ 16 MHz = 180 mA
   @ 20 MHz = 210 mA
   @ 25 MHz = 250 mA
4. This parameter is for inputs without pull ups or pull downs and 0V ≤ V_{IN} ≤ V_{CC}.
5. This parameter is for inputs with pull downs and V_{IH} = 2.4V.
6. This parameter is for inputs with pull ups and V_{IL} = 0.45V.
7. Not 100% tested.

A.C. SPECIFICATIONS

The A.C. Specifications given in the following tables consist of output delays, input setup requirements and input hold requirementts. All A.C. specifications are relative to the rising edge of the CLK signal.

A.C. specifications measurement is defined by the timing diagrams in the base Intel486 SX documentation (refer to the cover page of this document for the order number). The Boundary Scan (JTAG) timing waveforms are included in this document following the A.C. Specifications section.

16 MHz 3.3V Intel486™ SX Microprocessor A.C. Characteristics

V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Figure	Units	Notes
	Frequency	8	16		MHz	1X Clock
t_1	CLK Period	62.5	125	12.1	ns	
t_{1a}	CLK Period Stability		0.1%	12.1	ns	Adjacent Clocks
t_2	CLK High Time	20		12.1	ns	at 2V
t_3	CLK Low Time	20		12.1	ns	at 0.8V
t_4	CLK Fall Time		8	12.1	ns	2V to 0.8V
t_5	CLK Rise Time		8	12.1	ns	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Valid Delay	3	26	12.5	ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		42	12.6	ns	After Clock Edges(1)
t_8	PCHK# Valid Delay	3	35	12.4	ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	35	12.5	ns	
t_9	BLAST#, PLOCK# Float Delay		42	12.6	ns	After Clock Edges(1)
t_{10}	D0–D31, DP0–DP3 Write Data Valid Delay	3	30	12.5	ns	
t_{11}	D0–D31, DP0–DP3 Write Data Float Delay		42	12.6	ns	After Clock Edges(1)
t_{12}	EADS# Setup Time	12		12.2	ns	
t_{13}	EADS# Hold Time	3		12.2	ns	
t_{14}	KEN#, BS16#, BS8# Setup Time	12		12.2	ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		12.2	ns	
t_{16}	RDY#, BRDY# Setup Time	12		12.2	ns	

PRELIMINARY
SEE INTEL FOR DESIGN-IN
INFORMATION

16 MHz 3.3V Intel486™ SX Microprocessor A.C. Characteristics

V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified. (Continued)

Symbol	Parameter	Min	Max	Figure	Unit	Notes
t_{17}	RDY#, BRDY# Hold Time	3		12.2	ns	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	12		12.2	ns	
t_{19}	HOLD, AHOLD, BOFF# Setup Time	3		12.2	ns	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR Setup Time	14		12.2	ns	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR Hold Time	3		12.2	ns	
t_{22}	D0–D31 , DP0–DP3, A4–A31 Read Setup Time	10		12.2	ns	
t_{23}	D0–D31 , DP0–DP3, A4–A31 Read Hold Time	3		12.2	ns	
t_{24}	TCK Frequency		16	12.7	MHz	1X Clock
t_{25}	TCK Period	62.5		12.7	ns	(Note 3)
t_{26}	TCK High Time	10		12.7	ns	@2.0V
t_{27}	TCK Low Time	10		12.7	ns	@0.8V
t_{28}	TCK Rise Time		4	12.7	ns	(Note 2)
t_{29}	TCK Fall Time		4	12.7	ns	(Note 2)
t_{30}	TDI, TMS Setup Time	8		12.7	ns	(Note 4)
t_{31}	TDI, TMS Hold Time	10		12.7	ns	(Note 4)
t_{32}	TDO Valid Delay	3	25	12.7	ns	(Note 4)
t_{33}	TDO Float Delay		TBD	12.7	ns	(Note 4)
t_{34}	All Outputs (Non-Test Valid Delay)	3	25	12.7	ns	(Note 4)
t_{35}	All Outputs (Non-Test Float Delay)		36	12.7	ns	(Note 4)
t_{36}	All Inputs (Non-Test) Setup Time	8		12.7	ns	(Note 4)
t_{37}	All Inputs (Non-Test) Hold Time	10		12.7	ns	(Note 4)

PRELIMINARY
FOR DESIGN-IN
INFORMATION
SEE INTEL

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. Rise/Fall times are measured between 0.8V and 2.0V. Rise/Fall times can be relaxed by 1 ns per 10 ns increase in TCK period.
3. TCK period ≥ CLK period.
4. Parameter measured from TCK. Boundary scan tested at V_{CC} greater than or equal to 3.3V.

20 MHz 3.3V Intel486™ SX Microprocessor A.C. Characteristics

V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Figure	Unit	Notes
	Frequency	8	20		MHz	1X Clock
t_1	CLK Period	50	125	12.1	ns	
t_{1a}	CLK Period Stability		0.1%	12.1	ns	Adjacent Clocks
t_2	CLK High Time	16		12.1	ns	at 2V
t_3	CLK Low Time	16		12.1	ns	at 0.8V
t_4	CLK Fall Time		6	12.1	ns	2V to 0.8V
t_5	CLK Rise Time		6	12.1	ns	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Valid Delay	3	23	12.5	ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		37	12.6	ns	After Clock Edges(1)
t_8	PCHK# Valid Delay	3	28	12.4	ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	28	12.5	ns	
t_9	BLAST#, PLOCK# Float Delay		37	12.6	ns	After Clock Edges(1)
t_{10}	D0–D31 , DP0–DP3 Write Data Valid Delay	3	26	12.5	ns	
t_{11}	D0–D31 , DP0–DP3 Write Data Float Delay		37	12.6	ns	After Clock Edges(1)
t_{12}	EADS# Setup Time	10		12.2	ns	
t_{13}	EADS# Hold Time	3		12.2	ns	
t_{14}	KEN#, BS16#, BS8# Setup Time	10		12.2	ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		12.2	ns	
t_{16}	RDY#, BRDY# Setup Time	10		12.2	ns	
t_{17}	RDY#, BRDY# Hold Time	3		12.2	ns	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	12		12.2	ns	
t_{19}	HOLD, AHOLD, BOFF# Setup Time	3		12.2	ns	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR Setup Time	12		12.2	ns	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR Hold Time	3		12.2	ns	
t_{22}	D0–D31 , DP0–DP3, A4–A31 Read Setup Time	8		12.2	ns	
t_{23}	D0–D31 , DP0–DP3, A4–A31 Read Hold Time	3		12.2	ns	
t_{24}	TCK Frequency		20	12.7	MHz	1X Clock
t_{25}	TCK Period	50		12.7	ns	(Note 3)
t_{26}	TCK High Time	10		12.7	ns	@ 2.0V
t_{27}	TCK Low Time	10		12.7	ns	@0.8V
t_{28}	TCK Rise Time		4	12.7	ns	(Note 2)
t_{29}	TCK Fall Time		4	12.7	ns	(Note 2)

PRELIMINARY
SEE INTEL FOR DESIGN-IN
INFORMATION

20 MHz 3.3V Intel486™ SX Microprocessor A.C. Characteristics

V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified. (Continued)

Symbol	Parameter	Min	Max	Figure	Unit	Notes
t_{30}	TDI, TMS Setup Time	8		12.7	ns	(Note 4)
t_{31}	TDI, TMS Hold Time	10		12.7	ns	(Note 4)
t_{32}	TDO Valid Delay	3	25	12.7	ns	(Note 4)
t_{33}	TDO Float Delay		TBD	12.7	ns	(Note 4)
t_{34}	All Outputs (Non-Test Valid Delay)	3	25	12.7	ns	(Note 4)
t_{35}	All Outputs (Non-Test Float Delay)		36	12.7	ns	(Note 4)
t_{36}	All Inputs (Non-Test) Setup Time	8		12.7	ns	(Note 4)
t_{37}	All Inputs (Non-Test) Hold Time	10		12.7	ns	(Note 4)

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. Rise/Fall times are measured between 0.8V and 2.0V. Rise/Fall times can be relaxed by 1 ns per 10 ns increase in TCK period.
3. TCK period ≥ CLK period.
4. Parameter measured from TCK. Boundary scan tested at V_{CC} greater than or equal to 3.3V.

25 MHz 3.3V Intel486™ SX Microprocessor A.C. Characteristics

V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified.

Symbol	Parameter	Min	Max	Figure	Unit	Notes
	Frequency	8	25		MHz	1X Clock
t_1	CLK Period	40	125	12.1	ns	
t_{1a}	CLK Period Stability		0.1%	12.1	ns	Adjacent Clocks
t_2	CLK High Time	14		12.1	ns	at 2V
t_3	CLK Low Time	14		12.1	ns	at 0.8V
t_4	CLK Fall Time		4	12.1	ns	2V to 0.8V
t_5	CLK Rise Time		4	12.1	ns	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Valid Delay	3	20	12.5	ns	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, BREQ, HLDA Float Delay		28	12.6	ns	After Clock Edges(1)
t_8	PCHK# Valid Delay	3	24	12.4	ns	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	24	12.5	ns	
t_9	BLAST#, PLOCK# Float Delay		28	12.6	ns	After Clock Edges(1)
t_{10}	D0–D31 , DP0–DP3 Write Data Valid Delay	3	20	12.5	ns	
t_{11}	D0–D31, DP0–DP3 Write Data Float Delay		28	12.6	ns	After Clock Edges(1)
t_{12}	EADS# Setup Time	8		12.2	ns	
t_{13}	EADS# Hold Time	3		12.2	ns	

PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

2

25 MHz 3.3V Intel486™ SX Microprocessor A.C. Characteristics

V_{CC} = 3.3V ±0.3V; T_{CASE} = 0°C to +85°C; C_L = 50 pF unless otherwise specified. (Continued)

Symbol	Parameter	Min	Max	Figure	Unit	Notes
t_{14}	KEN#, BS16#, BS8# Setup Time	8		12.2	ns	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		12.2	ns	
t_{16}	RDY#, BRDY# Setup Time	8		12.2	ns	
t_{17}	RDY#, BRDY# Hold Time	3		12.2	ns	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	10		12.2	ns	
t_{19}	HOLD, AHOLD, BOFF# Setup Time	3		12.2	ns	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR Setup Time	10		12.2	ns	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR Hold Time	3		12.2	ns	
t_{22}	D0–D31, DP0–DP3, A4–A31 Read Setup Time	8		12.2	ns	
t_{23}	D0–D31, DP0–DP3, A4–A31 Read Hold Time	3		12.2	ns	
t_{24}	TCK Frequency		25	12.7	MHz	1X Clock
t_{25}	TCK Period	40		12.7	ns	(Note 3)
t_{26}	TCK High Time	10		12.7	ns	@2.0V
t_{27}	TCK Low Time	10		12.7	ns	@0.8V
t_{28}	TCK Rise Time		4	12.7	ns	(Note 2)
t_{29}	TCK Fall Time		4	12.7	ns	(Note 2)
t_{30}	TDI, TMS Setup Time	8		12.7	ns	(Note 4)
t_{31}	TDI, TMS Hold Time	10		12.7	ns	(Note 4)
t_{32}	TDO Valid Delay	3	25	12.7	ns	(Note 4)
t_{33}	TDO Float Delay		TBD	12.7	ns	(Note 4)
t_{34}	All Outputs (Non-Test Valid Delay)	3	25	12.7	ns	(Note 4)
t_{35}	All Outputs (Non-Test Float Delay)		36	12.7	ns	(Note 4)
t_{36}	All Inputs (Non-Test) Setup Time	8		12.7	ns	(Note 4)
t_{37}	All Inputs (Non-Test) Hold Time	10		12.7	ns	(Note 4)

PRELIMINARY
SEE INTEL FOR DESIGN-IN
INFORMATION

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. Rise/Fall times are measured between 0.8V and 2.0V. Rise/Fall times can be relaxed by 1 ns per 10 ns increase in TCK period.
3. TCK period ≥ CLK period.
4. Parameter measured from TCK. Boundary scan tested at V_{CC} greater than or equal to 3.3V.

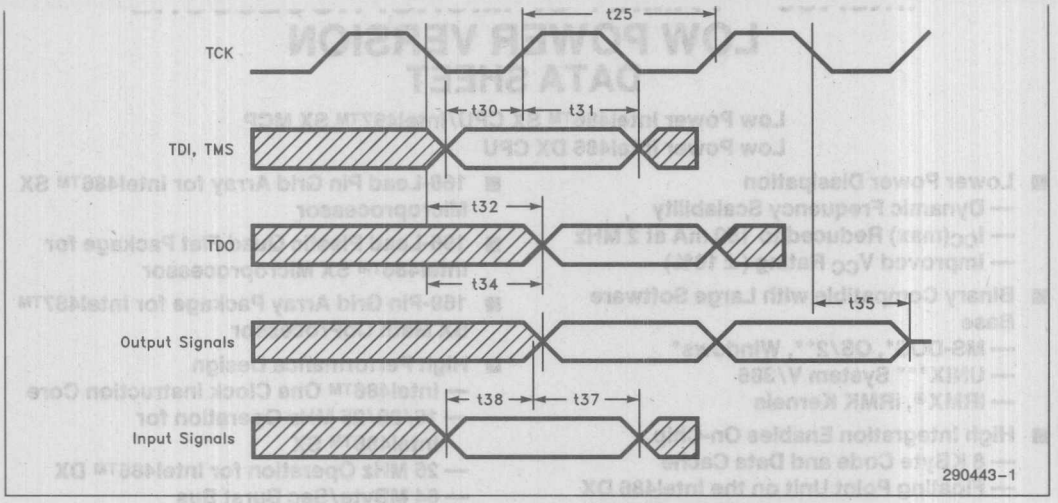


Figure 1. Test Signal Timing Diagram

Intel486™ FAMILY OF MICROPROCESSORS
LOW POWER VERSION
DATA SHEET

Low Power Intel486™ SX CPU/Intel487™ SX MCP
Low Power Intel486 DX CPU

- **Lower Power Dissipation**
  - — Dynamic Frequency Scalability
  - — I_{CC}(max) Reduced to 150 mA at 2 MHz
  - — Improved V_{CC} Rating (± 10%)
- **Binary Compatible with Large Software Base**
  - — MS-DOS*, OS/2**, Windows*
  - — UNIX*** System V/386
  - — iRMX®, iRMK Kernels
- **High Integration Enables On-Chip**
  - — 8 KByte Code and Data Cache
  - — Floating Point Unit on the Intel486 DX CPU and Intel487™ SX Math CoProcessor
  - — Paged, Virtual Memory Management
- **Easy to Use**
  - — Built-In Self Test
  - — Hardware Debugging Support
  - — Intel Software Support
  - — Extensive Third Party Software Support
- **168-Lead Pin Grid Array Package for Intel486 DX Microprocessor**
- **168-Lead Pin Grid Array for Intel486™ SX Microprocessor**
- **196-Lead Plastic Quad Flat Package for Intel486™ SX Microprocessor**
- **169-Pin Grid Array Package for Intel487™ SX Math CoProcessor**
- **High Performance Design**
  - — Intel486™ One Clock Instruction Core
  - — 16/20/25 MHz Operation for Intel486™ SX
  - — 25 MHz Operation for Intel486™ DX
  - — 64 MByte/Sec Burst Bus
  - — CHMOS IV Process Technology
  - — Dynamic Bus Sizing for 8-, 16- and 32-Bit Buses
- **Complete 32-Bit Architecture**
  - — Address and Data Buses
  - — Registers
  - — 8-, 16- and 32-Bit Data Types
- **Multiprocessor Support**
  - — Multiprocessor Instructions
  - — Cache Consistency Protocols
  - — Support for Second Level Cache

The data sheet describes both the Low Power Intel486 SX and the Low Power Intel486 DX microprocessors. The Intel487 SX Math CoProcessor will support the low power Intel486 SX microprocessor as an optional upgrade available through the retail channel.

The Low Power Intel486 family microprocessors meet today's need for high performance portables. Their combination of special features like dynamic frequency scaling, lower minimum frequency, improved V_{CC} operation and high integration contribute significantly to lower power dissipation and meet the needs of portable computing.

The Low Power capability is achieved by operating the Intel486 microprocessor in the 2X mode. The frequency can be varied dynamically between maximum to minimum as needed. The frequency change does not affect contents of the registers and data integrity is maintained. Power dissipation is reduced significantly at 2 MHz where I_{CC} is only 150 mA compared to 600 mA at 20 MHz. Low power versions are offered for both the Intel486 SX and the Intel486 DX microprocessors.

The Low Power Intel486 microprocessors are 100-percent compatible with all versions of the Intel386™ microprocessor family, assuring compatibility with the more than $40 billion software base of MS-DOS, Windows, OS/2 and UNIX/System operating system applications. The Low Power Intel486 microprocessor integrates the same RISC-technology, one clock per instruction integer core, on-chip cache, and memory management unit as the standard Intel486 microprocessor.

The Intel487 SX Math CoProcessor provides optional math upgrade capability for the Intel486 SX microprocessor and supports low power operation; providing end-users increased floating point performance for more than 2100 software packages that were designed to use Intel Math CoProcessors. Note that the Intel OverDrive™ Processor does not work in systems based on the Low Power Intel486 CPU.

*MS-DOS and Windows are trademarks of Microsoft Corp.
**OS/2 is a trademark of International Business Machines.
***UNIX is a trademark of UNIX Systems Laboratories.

December 1992
Order Number: 241199-002

Intel486™ Family of Microprocessors
Low Power Version
Data Sheet

CONTENTS PAGE

1.0 INTRODUCTION 2-818
1.1 Pinout 2-819
1.2 Pin Cross Reference (Intel486™ DX CPU) 2-824
1.3 Pin Cross Reference (Intel486™ SX CPU) 2-825
1.4 Pin Cross Reference (Intel486™ SX CPU PQFP Version) 2-826
1.5 Pin Description 2-827
1.6 Signal Description 2-832
1.7 Architecture Overview 2-835
1.8 Variable CPU Frequency 2-835

CONTENTS PAGE

2.0 D.C./A.C. SPECIFICATIONS 2-837
2.1 D.C. Specifications 2-837
2.2 Power Supply Current vs Frequency 2-840
2.3 A.C. Specifications 2-840

3.0 MATH UPGRADE FOR LOW POWER Intel486™ MICROPROCESSOR 2-846
3.1 Pinout 2-847
3.2 Pin Reference of Intel487™ SX Math CoProcessor 2-849
3.3 Intel487™ SX Math CoProcessor Pin Description 2-850

4.0 REVISION HISTORY 2-851

2

This document should be used in conjunction with the Intel486™ DX Microprocessor data sheet (order number 240440-004, June 1991) and the Intel486™ SX Microprocessor data sheet (order number 240950-002, October 1991).

1.0 INTRODUCTION

The Low Power Intel486 microprocessor brings Intel486 technology and performance to the portable computer market. The low power capability is achieved by a frequency scalability feature during normal operation. The operating frequency can be brought down dynamically resulting in lower power supply current (I_{CC}). This results in minimal power dissipation which ensures a longer battery life.

The Low Power Intel486 microprocessor integrates the same RISC-technology, one clock per instruction integer core, on-chip cache, and memory management unit as the standard Intel486 microprocessor.

The Low Power Intel486 microprocessor has the following special features:

- **Frequency Scalability**—This is achieved by operating the Intel486 microprocessor in the 2X clock mode. The frequency can be varied dynamically from maximum back to minimum or vice versa. The frequency change does not affect the register content of the CPU, thus data integrity is maintained.
- **Lower Minimum Frequency**—The Low Power Intel486 microprocessor can be operated at a minimum frequency of 2 MHz, at which I_{CC}(max) is only 150 mA, compared to an I_{CC}(max) of 600 mA at 20 MHz operation. The power dissipation is thus drastically reduced ensuring a longer battery life.
- **Improved V_{CC} Operation**—The Low Power Intel486 microprocessor has an improved V_{CC} rating of ±10%. Again this feature makes it extremely attractive to portable battery powered applications.

The above three features ensure power savings for portable computer systems resulting in prolonged battery life.

Besides the above special features, the Low Power Intel486 microprocessor has an identical feature set to the standard Intel486 CPU. This includes:

- **Binary Compatibility**—The Low Power Intel486 CPU is binary compatible with the 8086, 8088, 80186, 80286, i386™ SX, i386™ DX, Intel486™ SX and Intel486™ DX CPUs.
- **Full 32-Bit Integer Processor**—The Low Power Intel486 CPU performs a complete set of arithmetic and logical operations on 8-, 16-, and 32-bit data types using a full-width ALU and eight general-purpose registers.
- **Separate 32-Bit Address and Data Paths**—Four gigabytes of physical memory can be addressed directly.
- **Single-Cycle Execution**—Many instructions execute in a single clock cycle.
- **On-Chip Floating Point Unit**—This is available on the Intel486 DX CPU. The 32-, 64-, and 80-bit formats specified in IEEE standard 754 are supported. The unit is binary compatible with the 8087, 80287, i387™, i387™ SX, and Intel487™ math coprocessors and the Intel486™ CPU.
- **On-Chip Memory Management Unit**—Address-management and memory-space protection mechanisms maintain the integrity of memory. This is necessary in multitasking and virtual-memory environments, like those implemented by the UNIX and OS/2 operating systems. Both memory segmentation and paging are supported.
- **On-Chip Cache with Cache Consistency Support**—The internal write-through cache can hold 8 KBytes of data or instructions. Cache hits are as fast as read accesses to a processor register. Bus activity is tracked to detect alterations in the memory which internal cache represents. The internal cache can be invalidated or flushed, so that an external cache controller can maintain cache consistency in multi-processor environments.
- **External Cache Control**—Write-back and flush controls over an external cache are provided so that the processor can maintain cache consistency in multi-processor environments.
- **Instruction Pipelining**—The fetching, decoding, execution and address translation of instructions are overlaped within the Low Power Intel486 microprocessor. This results in a continuous execution rate of one clock cycle per instruction, for most instructions.
- **Burst Cycles**—Burst transfers allow a new doubleword to be read from memory each clock cycle. With this capability the internal cache and instruction prefetch buffer can be filled very rapidly.
- **Write Buffers**—The processor contains write buffers to enhance the performance of consecutive writes to memory. The Low Power Intel486 CPU can continue operations internally after a write, without waiting for the write to be executed on the external bus.
- **Bus Backoff**—If another bus master needs control of the bus during a Low Power Intel486 microprocessor initiated bus cycle, the Low Power

Intel486 microprocessor will float its bus signals, then restart its cycle when the bus becomes available again.

- **Instruction Restart**—Programs can continue execution following an exception generated by an unsuccessful attempt to access memory. This feature is important for supporting demand-paged virtual memory applications.
- **Dynamic Bus Sizing**—External controllers can dynamically alter the effective width of the data bus. Bus widths of 8, 16 or 32 bits can be used.

1.1 Pinout

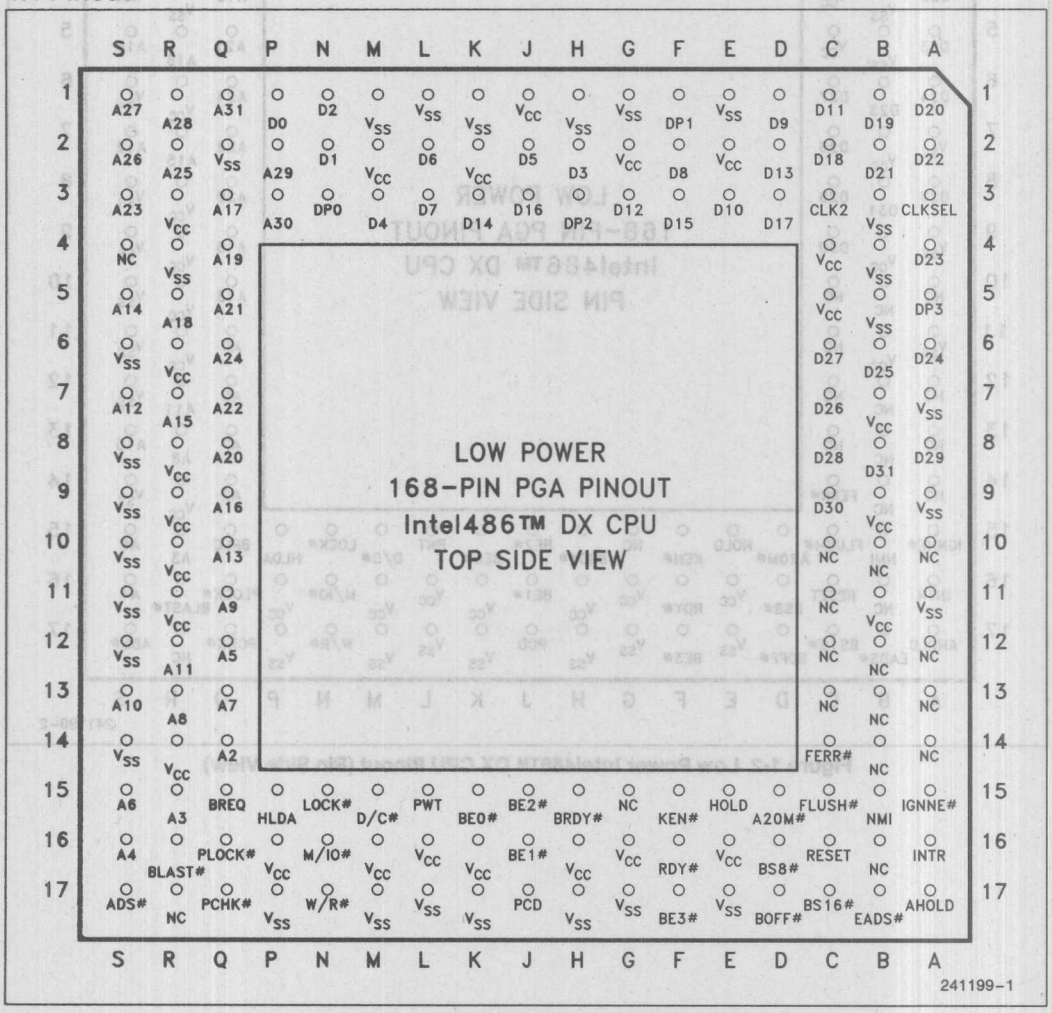


Figure 1-1. Low Power Intel486™ DX CPU Pinout (Top Side View)

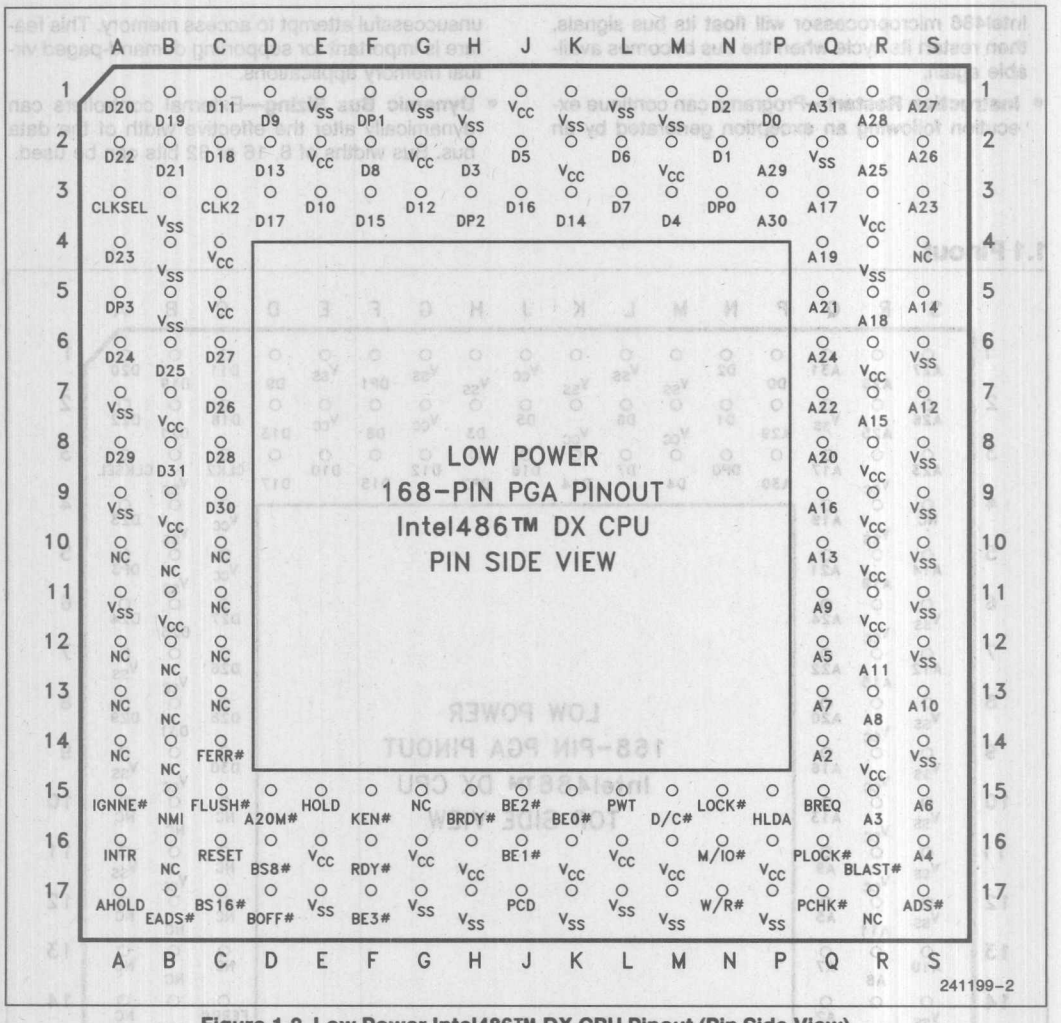


Figure 1-2. Low Power Intel486™ DX CPU Pinout (Pin Side View)

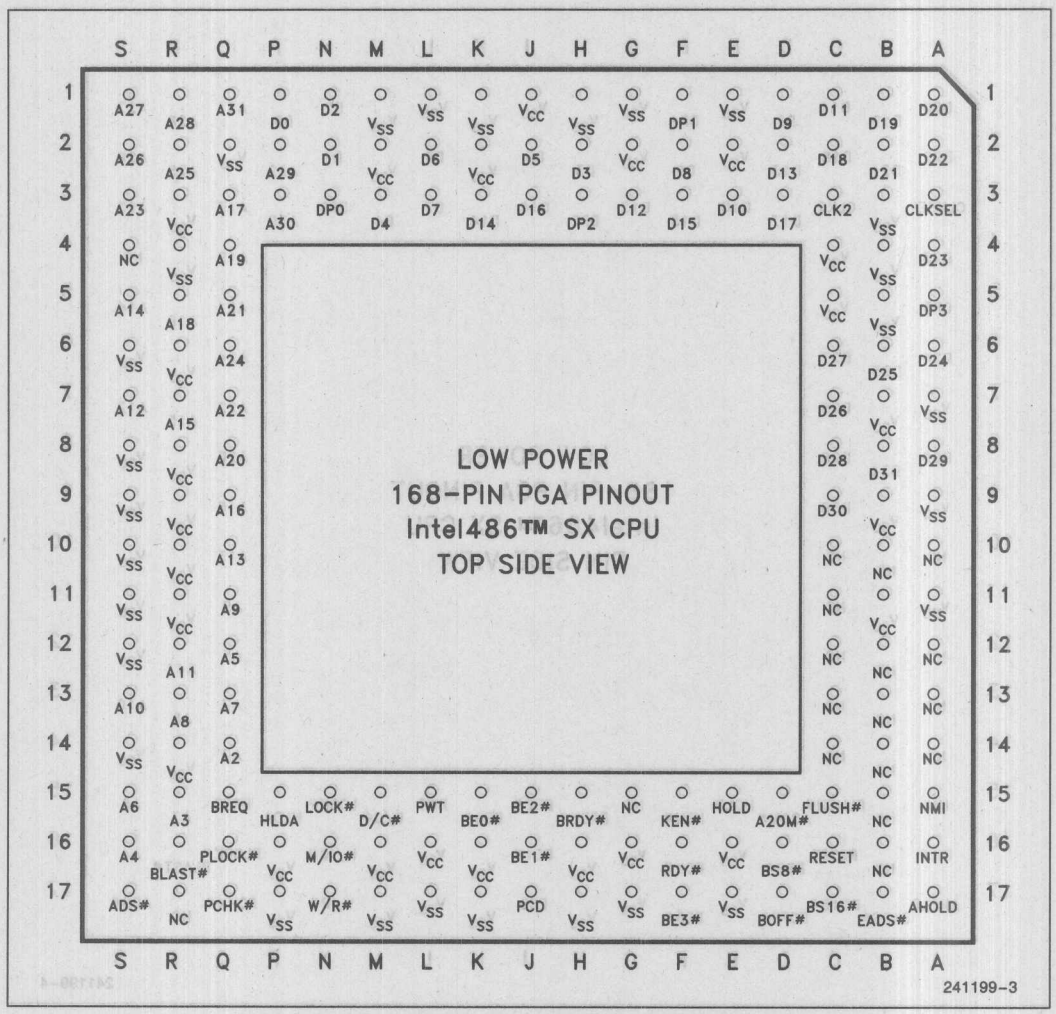


Figure 1-3. Low Power Intel486™ SX CPU Pinout (Top Side View)

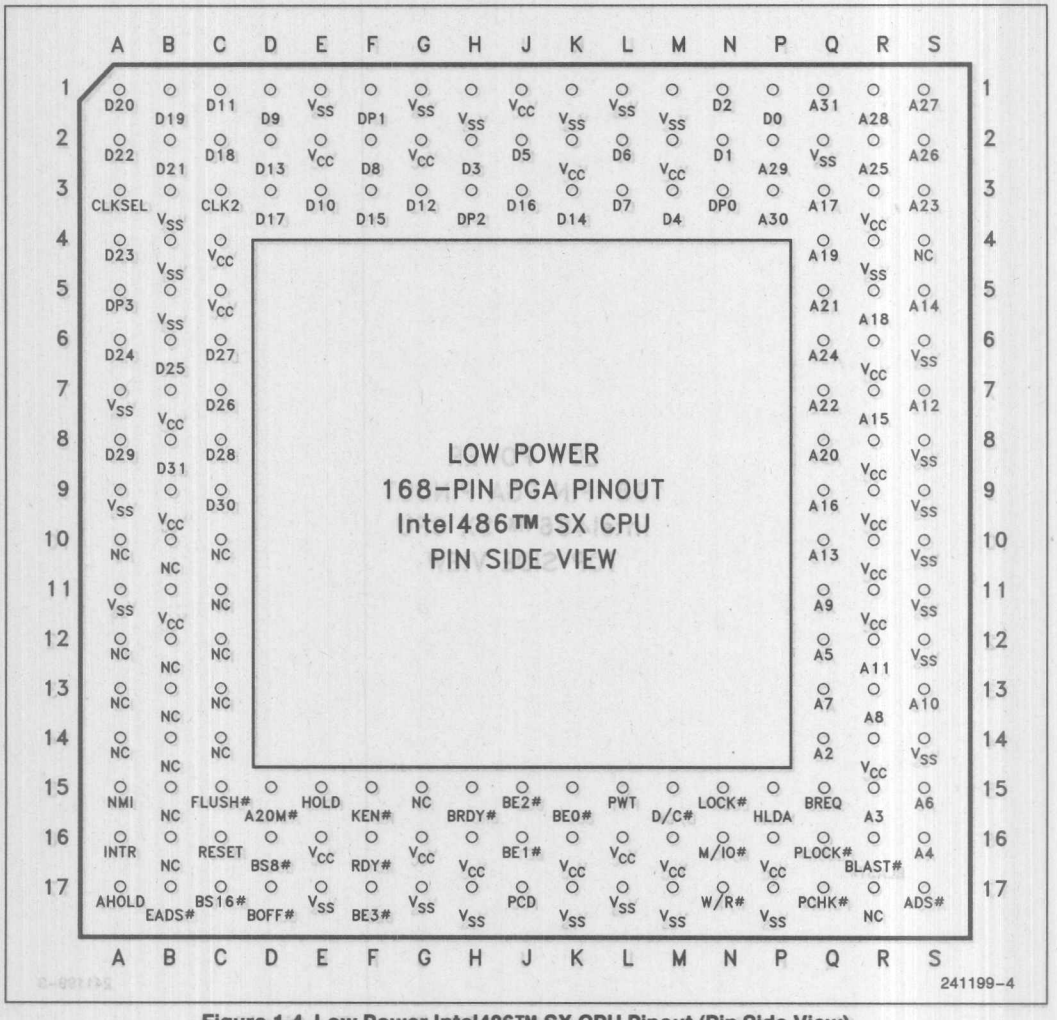


Figure 1-4. Low Power Intel486™ SX CPU Pinout (Pin Side View)



Figure 1-5. Low Power Intel486™ SX CPU 196-Lead PQFP Pinout

1.2 Pin Cross Reference (Intel486™ DX CPU)

Address		Data		Control		N/C	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	A10	B7	A7
A_3	R15	D_1	N2	ADS#	S17	A12	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	A13	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	A14	C4	B3
A_6	S15	D_4	M3	BE1#	J16	B10	C5	B4
A_7	Q13	D_5	J2	BE2#	J15	B12	E2	B5
A_8	R13	D_6	L2	BE3#	F17	B13	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16	B14	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17	B16	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15	C10	H16	G17
A_{12}	S7	D_{10}	E3	BREQ	Q15	C11	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16	C12	K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17	C13	K16	K1
A_{15}	R7	D_{13}	D2	CLK2	C3	G15	L16	K17
A_{16}	Q9	D_{14}	K3	CLKSEL	A3	R17	M2	L1
A_{17}	Q3	D_{15}	F3	D/C#	M15	S4	M16	L17
A_{18}	R5	D_{16}	J3	DP0	N3		P16	M1
A_{19}	Q4	D_{17}	D3	DP1	F1		R3	M17
A_{20}	Q8	D_{18}	C2	DP2	H3		R6	P17
A_{21}	Q5	D_{19}	B1	DP3	A5		R8	Q2
A_{22}	Q7	D_{20}	A1	EADS#	B17		R9	R4
A_{23}	S3	D_{21}	B2	FERR#	C14		R10	S6
A_{24}	Q6	D_{22}	A2	FLUSH#	C15		R11	S8
A_{25}	R2	D_{23}	A4	HLDA	P15		R14	S9
A_{26}	S2	D_{24}	A6	HOLD	E15			S10
A_{27}	S1	D_{25}	B6	IGNNE#	A15			S11
A_{28}	R1	D_{26}	C7	INTR	A16			S12
A_{29}	P2	D_{27}	C6	KEN#	F15			S14
A_{30}	P3	D_{28}	C8	LOCK#	N15			
A_{31}	Q1	D_{29}	A8	M/IO#	N16			
		D_{30}	C9	NMI	B15			
		D_{31}	B8	PCD	J17			
				PCHK#	Q17			
				PWT	L15			
				PLOCK#	Q16			
				RDY#	F16			
				RESET	C16			
				W/R#	N17			

1.3 Pin Cross Reference (Intel486™ SX CPU)

Address		Data		Control		N/C	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	A10	B7	A7
A_3	R15	D_1	N2	ADS#	S17	A12	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	A13	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	A14	C4	B3
A_6	S15	D_4	M3	BE1#	J16	B10	C5	B4
A_7	Q13	D_5	J2	BE2#	J15	B12	E2	B5
A_8	R13	D_6	L2	BE3#	F17	B13	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16	B14	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17	B15	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15	B16	H16	G17
A_{12}	S7	D_{10}	E3	BREQ	Q15	C10	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16	C11	K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17	C12	K16	K1
A_{15}	R7	D_{13}	D2	CLK2	C3	C13	L16	K17
A_{16}	Q9	D_{14}	K3	CLKSEL	A3	C14	M2	L1
A_{17}	Q3	D_{15}	F3	D/C#	M15	G15	M16	L17
A_{18}	R5	D_{16}	J3	DP0	N3	R17	P16	M1
A_{19}	Q4	D_{17}	D3	DP1	F1	S4	R3	M17
A_{20}	Q8	D_{18}	C2	DP2	H3		R6	P17
A_{21}	Q5	D_{19}	B1	DP3	A5		R8	Q2
A_{22}	Q7	D_{20}	A1	EADS#	B17		R9	R4
A_{23}	S3	D_{21}	B2	FLUSH#	C15		R10	S6
A_{24}	Q6	D_{22}	A2	HLDA	P15		R11	S8
A_{25}	R2	D_{23}	A4	HOLD	E15		R14	S9
A_{26}	S2	D_{24}	A6	INTR	A16			S10
A_{27}	S1	D_{25}	B6	KEN#	F15			S11
A_{28}	R1	D_{26}	C7	LOCK#	N15			S12
A_{29}	P2	D_{27}	C6	M/IO#	N16			S14
A_{30}	P3	D_{28}	C8	NMI	B15			
A_{31}	Q1	D_{29}	A8	PCD	J17			
		D_{30}	C9	PCHK#	Q17			
		D_{31}	B8	PWT	L15			
				PLOCK#	Q16			
				RDY#	F16			
				RESET	C16			
				W/R#	N17			

2

1.4 Pin Cross Reference by Signal Type (Intel486™ SX PQFP CPU)

Address		Data		Control		N/C	V_{CC}	V_{SS}
A_2	146	D_0	17	A20M#	104	15	6	1
A_3	150	D_1	18	ADS#	145	34	19	11
A_4	152	D_2	20	AHOLD	129	52	24	21
A_5	154	D_3	23	BE0#	117	56	28	22
A_6	158	D_4	25	BE1#	116	60	36	33
A_7	159	D_5	26	BE2#	115	64	49	40
A_8	161	D_6	27	BE3#	113	68	54	50
A_9	163	D_7	29	BLAST#	144	72	62	58
A_{10}	165	D_8	31	BOFF#	137	73	70	66
A_{11}	172	D_9	32	BRDY#	138	75	84	86
A_{12}	174	D_{10}	35	BREQ	118	76	93	95
A_{13}	176	D_{11}	37	BS8#	135	77	98	96
A_{14}	178	D_{12}	38	BS16#	136	78	107	99
A_{15}	180	D_{13}	39	CLK2	123	79	112	109
A_{16}	181	D_{14}	41	CLKSEL	127	81	119	114
A_{17}	183	D_{15}	42	D/C#	110	82	125	121
A_{18}	189	D_{16}	44	DP0	16	83	131	126
A_{19}	191	D_{17}	45	DP1	30	85	147	141
A_{20}	193	D_{18}	46	DP2	43	87	164	148
A_{21}	2	D_{19}	47	DP3	57	88	170	167
A_{22}	3	D_{20}	48	EADS#	105	89	175	168
A_{23}	4	D_{21}	51	FLUSH#	102	90	179	177
A_{24}	5	D_{22}	53	HLDA	122	91	184	182
A_{25}	7	D_{23}	55	HOLD	130	92	196	194
A_{26}	8	D_{24}	59	INTR	101	94		
A_{27}	9	D_{25}	61	KEN#	132	97		
A_{28}	10	D_{26}	63	LOCK#	142	124		
A_{29}	12	D_{27}	65	M/IO#	111	134		
A_{30}	13	D_{28}	67	NMI	100	140		
A_{31}	14	D_{29}	69	PCD	106	149		
		D_{30}	71	PCHK#	139	151		
		D_{31}	74	PWT	108	153		
				PLOCK#	143	155		
				RDY#	133	157		
				RESET	103	160		
				TCK	128	162		
				TDI	185	166		
				TDO	80	166		
				TMS	187	169		
				UP#	156	171		
				W/R#	120	173		
						186		
						188		
						190		
						192		
						195		

1.5 Pin Description

The following table provides brief pin descriptions.

Symbol	Type	Name and Function
CLK2	I	**CLK2** provides the fundamental timing for the Low Power Intel486 microprocessor. This is twice the internal frequency of the CPU.
CLKSEL	I	**Clock Select** pin selects the 2X mode required for the Low Power Intel486 CPU. A well defined pulse on this pin establishes the phase relationship of the 2X clock. With the exception of a pulse during cold reset, this pin should be driven low all the time and must be free of spikes or glitches.
ADDRESS BUS		
A31–A4 A3–A2	I/O O	A31–A2 are the **address lines** of the microprocessor. A31–A2, together with the byte enables BE0#–BE3#, define the physical area of memory or input/output space accessed. Address lines A31–A4 are used to drive addresses into the microprocessor to perform cache line invalidations. Input signals must meet setup and hold times t_{22} and t_{23}. A31–A2 are not driven during bus or address hold.
BE0–3#	O	The **byte enable** signals indicate active bytes during read and write cycles. During the first cycle of a cache fill, the external system should assume that all byte enables are active. BE3# applies to D24–D31, BE2# applies to D16–D23, BE1# applies to D8–D15 and BE0# applies to D0–D7. BE0#–BE3# are active LOW and are not driven during bus hold.
DATA BUS		
D31–D0	I/O	These are the **data lines** for the Low Power Intel486 micrprocessor. Lines D0–D7 define the least significant byte of the data bus while lines D24–D31 define the most significant byte of the data bus. These signals must meet setup and hold times t_{22} and t_{23} for proper operation on reads. These pins are driven during the second and subsequent clocks of write cycles.
DATA PARITY		
DP0–DP3	I/O	There is one **data parity** pin for each byte of the data bus. Data parity is generated on all write data cycles with the same timing as the data driven by the Low Power Intel486 microprocessor. Even parity information must be driven back into the microprocessor on the data parity pins with the same timing as read information to insure that the correct parity check status is indicated by the Low Power Intel486 microprocessor. The signals read on these pins do not affect program execution. Input signals must meet setup and hold times t_{22} and t_{23}. DP0–DP3 should be connected to V_{CC} through a pullup resistor in systems which do not use parity. DP0–DP3 are active HIGH and are driven during the second and subsequent clocks of write cycles.
PCHK#	O	**Parity Status** is driven on the PCHK# pin the clock after ready for read operations. The parity status is for data sampled at the end of the previous clock. A parity error is indicated by PCHK# being LOW. Parity status is only checked for enabled bytes as indicated by the byte enable and bus size signals. PCHK# is valid only in the clock immediately after read data is returned to the microprocessor. At all other times PCHK# is inactive (HIGH). PCHK# is never floated.

1.5 Pin Description (Continued)

Symbol	Type	Name and Function
BUS CYCLE DEFINITION		
M/IO# D/C# W/R#	O O O	The **memory/input-output, data/control** and **write/read** lines are the primary bus definitions signals. These signals are driven valid as the ADS# signal is asserted.

M/IO#	D/C#	W/R#	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Halt/Special Cycle
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Code Read
1	0	1	Reserved
1	1	0	Memory Read
1	1	1	Memory Write

The bus definition signals are not driven during bus hold and follow the timing of the address bus.

LOCK#	O	The **bus lock** pin indicates that the current bus cycle is locked. The Low Power Intel486 microprocessor will not allow a bus hold when LOCK# is asserted (but address holds are allowed). LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after the last clock of the last locked bus cycle. The last locked cycle ends when ready is returned. LOCK# is active LOW and is not driven during bus hold. Locked read cycles will not be transformed into cache fill cycles if KEN# is returned active.
PLOCK#	O	The **pseudo-lock#** pin indicates that the current bus transaction requires more than one bus cycle to complete. Examples of such operations are segment table reads (64 bits), cache line fills (128 bits). The Low Power Intel486 microprocessor will drive PLOCK# active until the addresses for the last bus cycle of the transaction have been driven regardless of whether RDY# or BRDY# have been returned. Normally PLOCK# and BLAST# are inverse of each other. PLOCK# is a function of the BS8#, BS16# and KEN# inputs. PLOCK# should be sampled only if the clock ready is returned. PLOCK# is active LOW and is not driven during bus hold.
BUS CONTROL		
ADS#	O	The **address status** output indicates that a valid bus cycle definition and address are available on the cycle definition lines and address bus. ADS# is driven active in the same clock as the addresses are driven. ADS# is active LOW and is not driven during bus hold.
RDY#	I	The **non-burst Ready** input indicates that the current bus cycle is complete. RDY# indicates that the external system has presented valid data in response to a read or that the external system has accepted data in response to a write. RDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle. RDY# is active during address hold. Data can be returned to the processor while AHOLD is active. RDY# is active LOW and is provided with a small pullup resistor. RDY# must satisfy the setup and hold times t_{16} and t_{17} for proper chip operation.

1.5 Pin Description (Continued)

Symbol	Type	Name and Function
BURST CONTROL		
BRDY#	I	The **burst ready** input performs the same function during a burst cycle that RDY# performs during a non-burst cycle. BRDY# indicates that the external system has presented valid data in response to a read for that the external system has accepted data in response to a write. BRDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle. BRDY# is sampled in the second and subsequent clocks of a burst cycle. The data presented on the data bus will be strobed into the microprocessor when BRDY# is sampled active. If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is prematurely aborted. BRDY# is active LOW and is provided with a small pullup resistor. BRDY# must satisfy the setup and hold times t_{16} and t_{17}.
BLAST#	O	The **burst last** signal indicates that the next time BRDY# is returned the burst bus cycle is complete. BLAST# is active for both burst and non-burst bus cycles. BLAST# is active LOW and is not driven during bus hold.
INTERRUPTS		
RESET	I	The **reset** input forces the Low Power Intel486 microprocessor to begin execution at a known state. The microprocessor cannot begin execution of instructions until at least 1 ms after V_{CC} and CLK2 have reached their proper D.C. and A.C. specifications. The RESET pin should remain active during this time to insure proper microprocessor operation. However, for warm boot-ups RESET should remain active for at least 30 CLK2 periods. RESET is active HIGH. RESET is asynchronous but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
INTR	I	The **maskable interrupt** indicates that an external interrupt has been generated. If the internal interrupt flag is set in EFLAGS, active interrupt processing will be initiated. The Low Power Intel486 microprocessor will generate two locked interrupt acknowledge bus cycles in response to the INTR pin going active. INTR must remain active until the interrupt acknowledges have been performed to assure that the interrupt is recognized. INTR is active HIGH and is not provided with an internal pulldown resistor. INTR is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
NMI	I	The **non-maskable interrupt** request signal indicates that an external non-maskable interrupt has been generated. NMI is rising edge sensitive. NMI must be held LOW for at least eight CLK2 periods before this rising edge. NMI is not provided with an internal pulldown resistor. NMI is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
BUS ARBITRATION		
BREQ	O	The **internal cycle pending** signal indicates that the Low Power Intel486 microprocessor has internally generated a bus request. BREQ is generated whether or not the Low Power Intel486 microprocessor is driving the bus. BREQ is active high and is never floated.
HOLD	I	The **bus hold request** allows another bus master complete control of the Low Power Intel486 microprocessor bus. In response to HOLD going active the Low Power Intel486 microprocessor will float most of its output and input/output pins HLDA will be asserted after completing the current bus cycle, burst cycle or sequence of locked cycles. The Low Power Intel486 microprocessor will remain in this state until HOLD is deasserted. HOLD is active high and is not provided with an internal pulldown resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper operation.

1.5 Pin Description (Continued)

Symbol	Type	Name and Function
BUS ARBITRATION (Continued)		
HLDA	O	**Hold acknowledge** goes active in response to a hold request presented on the HOLD pin. HDLA indicates that the Low Power Intel486 microprocessor has given the bus to another local bus master. HLDA is driven active in the same clock that the Low Power Intel486 microprocessor floats its bus. HLDA is driven inactive when leaving bus hold. HLDA is active HIGH and remains driven during bus hold.
BOFF#	I	The **backoff** input forces the Low Power Intel486 microprocessor to float its bus in the next clock. The microprocessor will float all pins normally floated during hold but HLDA will not be asserted in response to BOFF#. BOFF# has higher priority than RDY# or BRDY#; if both are returned in the same clock, BOFF# takes effect. The microprocessor remains in bus hold until BOFF# is negated. If a bus cycle was in progress when BOFF# was asserted the cycle will be restarted. BOFF# is active LOW and must meet setup and hold times t_{18} and t_{19} for proper operation.
CACH INVALIDATION		
AHOLD	I	The **address** hold request allows another bus master access to the Low Power Intel486 microprocessor's address bus for a cache invalidation cycle. The Low Power Intel486 microprocessor will stop driving its address bus in the clock following AHOLD going active. Only the address bus will be floated during address hold, the remainder of the bus will remain active. AHOLD is active HIGH and is provided with a small internal pulldown resistor. For proper operation AHOLD must meet setup and hold times t_{18} and t_{19}.
EADS#	I	This signal indicates that a valid **external address** has been driven onto the Low Power Intel486 microprocessor address pins. This address will be used to perform an internal cache invalidation cycle. EADS# is active LOW and is provided with an internal pullup resistor. EADS# must satisfy setup and hold times t_{12} and t_{13} for proper operation.
CACHE CONTROL		
KEN#	I	The **cache enable** pin is used to determine whether the current cycle is cacheable. When the Low Power Intel486 microprocessor generates a cycle that can be cached and KEN# is active, the cycle will become a cache line fill cycle. Returning KEN# active one clock before ready during the last read in the cache line fill will cause the line to be placed in the on-chip cache. KEN# is active LOW and is provided with a small internal pull-up resistor. KEN# must satisfy setup and hold times t_{14} and t_{15} for proper operation.
FLUSH#	I	The **cache flush** input forces the Low Power Intel486 microprocessor to flush its entire internal cache. FLUSH# is active low and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition in any specific clock. FLUSH# being sampled low in the clock before the falling edge of RESET causes the Low Power Intel486 microprocessor to enter the tri-state test mode.
PAGE CACHEABILITY		
PWT PCD	O O	The **page write-through** and **page cache disable** pins reflect the state of the page attribute bits, PWT and PCD, in the page table entry, page directory entry or control register 3 (CR3), when paging is enabled. If paging is disabled, the CPU ignores the PCD and PWT bits and assumes they are zero for the purpose of caching and driving PCD and PWT pins. PWT and PCD have the same timing as the cycle definition pins (M/IO#, D/C#, and W/R#). PWT and PCD are active HIGH and are not driven during bus hold. PCD is masked by the cache disable bit (CD) in Control Register 0.

1.5 Pin Description (Continued)

Symbol	Type	Name and Function
NUMERIC ERROR REPORTING (Present on Intel486 DX CPU Only)		
FERR#	O	The **floating point error** pin is driven active when a floating point error occurs. FERR# is similar to the ERROR# pin on the i387 Math CoProcessor. FERR# is included for compatibility with systems using DOS type floating point error reporting. FERR# is active LOW, and is not floated during bus hold.
IGNNE#	I	When the **ignore numeric error** pin is asserted, the Low Power Intel486 microprocessor will ignore a numeric error and continue executing non-control floating point instructions. When IGNNE# is deasserted the Low Power Intel486 microprocessor will freeze on a non-control floating point instruction, if a previous floating point instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set. IGNNE# is active LOW and is provided with a small internal pullup resistor. IGNNE# is asynchronous but setup and hold times t_{20} and t_{21} must be met to insure recognition on any specific clock.
BUS SIZE CONTROL		
BS16# BS8#	I I	The **bus size 16** and **bus size 8** pins (bus sizing pins) cause the Low Power Intel486 microprocessor to run multiple bus cycles to complete a request from devices that cannot provide or accept 32 bits of data in a single cycle. The bus sizing pins are sampled every clock. The state of these pins in the clock before ready is used by the Low Power Intel486 microprocessor to determine the bus size. These signals are active LOW and are provided with internal pullup resistors. These inputs must satisfy setup and hold times t_{14} and t_{15} for proper operation.
ADDRESS MASK		
A20M#	I	When the **address bit 20 mask** pin is asserted, the Low Power Intel486 microprocessor masks physical address bit 20 (A20) before performing a lookup to the internal cache or driving a memory cycle on the bus. A20M# emulates the address wraparound at one Mbyte which occurs on the 8086. A20M# is active LOW and should be asserted only when the processor is in real mode. This pin is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For proper operation , A20M# should be sampled high at the falling edge of RESET.
PERFORMANCE UPGRADE SUPPORT (Intel486 SX CPU PQFP Version Only)		
UP#	I	*Upgrade Present* forces the Intel486 SX CPU to tri-state all its outputs and enter the power down mode. UP# is active low and is sampled at all times, including after power-up and during reset.
TEST ACCESS PORT (Intel486 SX CPU PQFP Version Only)		
TCK	I	*Test Clock* is an input to the Intel486™ CPU and provides the clocking function required by the JTAG Boundary scan feature. TCK is used to clock state information and data into component on the rising edge of TCK on TMS and TDI, respectively. Data is clocked out of the part on the falling edge of TCK and TDO.
TDI	I	*Test Data Input* is the serial input used to shift JTAG instructions and data into component. TDI is sampled on the rising edge of TCK, during the SHIFT-IR and SHIFT-DR TAP controller states. During all other tap controller states, TDI is a "don't care".
TDO	O	*Test Data Output* is the serial output used to shift JTAG instructions and data out of the component. TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT-DR TAP controller states. At all other times TDO is driven to the high impedance state.
TMS	I	*Test Mode Select* is decoded by the JTAG TAP (Tap Access Port) to select the operation of the test logic. TMS is sampled on the rising edge of TCK. To guarantee deterministic behavior of the TAP controller TMS is provided with an internal pull-up resistor.

OUTPUT PINS

Table 1-1. lists all the output pins, indicating their active level, and when they are floated.

Table 1-1. Output Pins

Name	Active Level	When Floated
BREQ	HIGH	
HLDA	HIGH	
BE0#–BE3#	LOW	Bus Hold
PWT, PCD	HIGH	Bus Hold
W/R#, D/C#, M/IO#	HIGH/LOW	Bus Hold
LOCK#	LOW	Bus Hold
PLOCK#	LOW	Bus Hold
ADS#	LOW	Bus Hold
BLAST#	LOW	Bus Hold
PCHK#	LOW	
FERR#*	LOW	
A2-A3	HIGH	Bus, Address Hold

*Present on Intel486 DX CPU Only

INPUT PINS

Table 1-2 lists all input pins, indicating their active level, and whether they are synchronous or asynchronous inputs.

Table 1-2. Input Pins

Name	Active Level	Synchronous/ Asynchronous
CLK2		
CLKSEL		
RESET	HIGH	Asynchronous
HOLD	HIGH	Synchronous
AHOLD	HIGH	Synchronous
EADS#	LOW	Synchronous
BOFF#	LOW	Synchronous
FLUSH#	LOW	Asynchronous
A20M#	LOW	Asynchronous
BS16#, BS8#	LOW	Synchronous
KEN#	LOW	Synchronous
RDY#	LOW	Synchronous
BRDY#	LOW	Synchronous
INTR	HIGH	Asynchronous
NMI	HIGH	Asynchronous
IGNNE#*(1)	LOW	Asynchronous
UP#(2)	LOW	Asynchronous

NOTES:
1. The IGNNE# pin is present on the Intel486 DX CPU and Intel487 SX MCP only.
2. The UP# pin is present on the Intel486 SX CPU PQFP package only.

INPUT/OUTPUT PINS

Table 1-3 lists all the input/output pins, indicating their active level and when they are floated.

Table 1-3. Input/Output Pins

Name	Active Level	When Floated
D0–D31	HIGH	Bus Hold
DP0–DP3	HIGH	Bus Hold
A4–A31	HIGH	Bus, Address Hold

Table 1-4. Test Pins

Name	Input or Output	Sampled/ Driven On
TCK	Input	N/A
TDI	Input	Rising Edge of TCK
TDO	Output	Falling Edge of TCK
TMS	Input	Rising Edge of TCK

Table 1-5. Component and Revision ID (PGA)

i486 SX Microprocessor/ i487 SX Math CoProcessor Stepping Name	Component ID	Revision ID
A0	04	20

NOTE:
Table 1-5 shows the Component ID number and Revision ID number for the A-0 stepping of the Intel486 SX Microprocesor and Intel487 SX Math CoProcessor. When an Intel487 SX Math CoProcessor is installed in the system, the Component ID and Revision ID is provided by the Intel487 SX Math CoProcessor and not the Intel486 SX Microprocessor. The Component ID and Revision ID read by the BIOS/software may change when a Performance Upgrade Component, such as the Intel487 SX Math CoProcessor, is installed in an Intel486 SX Microprocessor based system.

1.6 Signal Description

With the exception of CLK2 and CLKSEL, all signals follow the same definition as the Intel486 microprocessor. The A.C. timing parameters for all of these signals are given in Table 2-7.

CLOCK (CLK2)

CLK2 provides the fundamental timing for the Low Power Intel486 microprocessor. It is divided by two internally to generate the internal processor clock used for instruction execution. The internal clock is comprised of two phases, "phase one" and "phase two". Each CLK2 period is a phase of the internal clock. Figure 1-5 illustrates the relationship. If de-

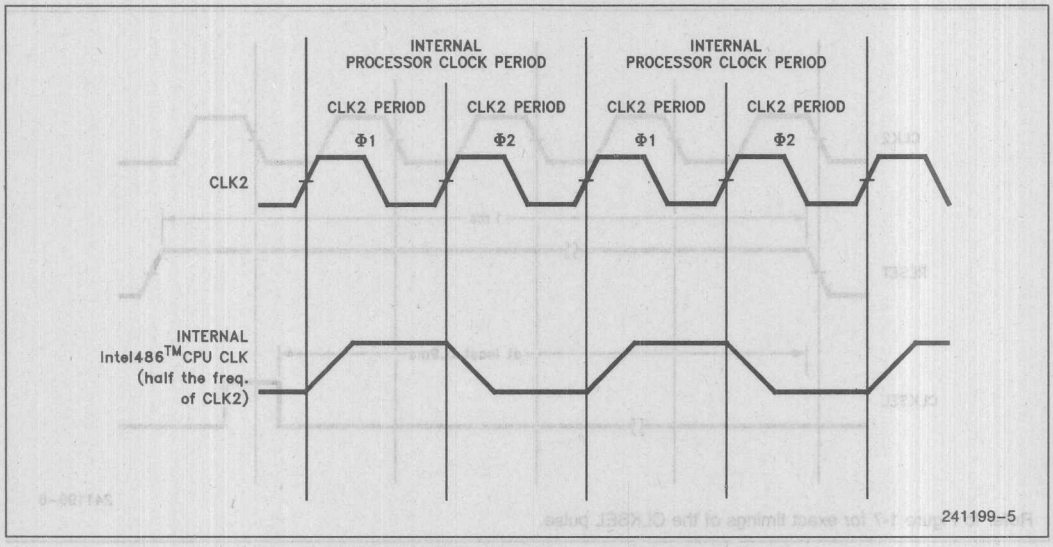


Figure 1-5. CLK2 Signal and Internal Processor Clock

sired, the phase of the internal processor clock can be synchronized to a known phase by ensuring the pulse on the CLKSEL pin meets the applicable timings during cold boot (power-up reset).

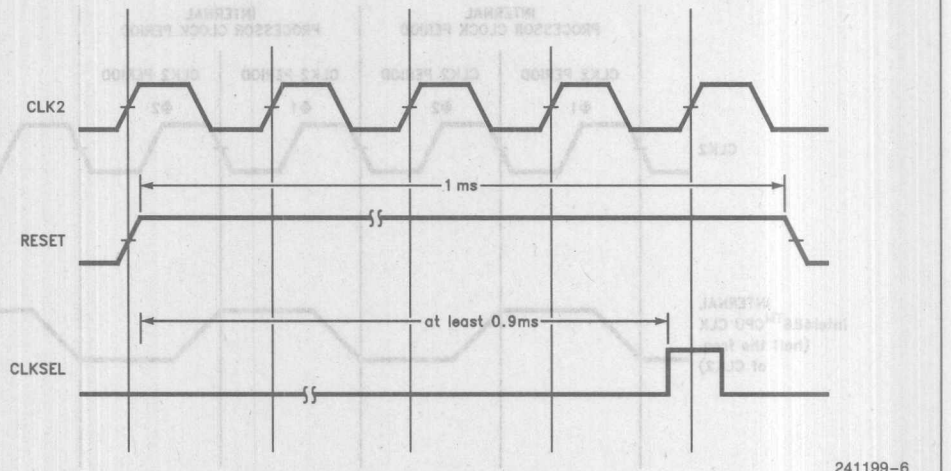
All set-up, hold, float-delay and valid delay timings are referenced to the phase one of the clock.

The internal processor clock (CLK) is similar to the clock signal of the standard Intel486 microprocessor. All I/O signals get sampled on the rising edge of this signal, i.e. the rising edge of phase one. Thus it is important to synchronize the external circuitry with the phase one of CLK2.

CLKSEL

Clock Select pin selects the 2X mode required for the Low Power Intel486 CPU. This pin should be driven low after power-up and during the entire operation of the CPU. However, a well defined pulse is required on CLKSEL pin during cold boot (power-up reset) to establish the phase relationship of the 2X clock. The reset pulse width during cold reset should be at least 1 ms. As shown in Figure 1-6, the pulse on CLKSEL should be asserted by the end of reset (approximately 0.9 ms after driving reset active) and at least 30 CLK2 periods before the falling edge of reset.

Figure 1-7 shows the detailed timing definition of this pulse. The pulse on CLKSEL pin is only required during power-up reset. During all other times including warm resets the CLKSEL pin should be driven low and must be free of spikes or glitches. After the power-up reset, the system must track the phase of CLK2 at all times including during warm resets so that the input/output signals can be sampled at the appropriate clock edge. The phase relationship is described in the next section.



Refer to Figure 1-7 for exact timings of the CLKSEL pulse.

Figure 1-6. CLKSEL Pulse with Reference to the Reset Pulse Width

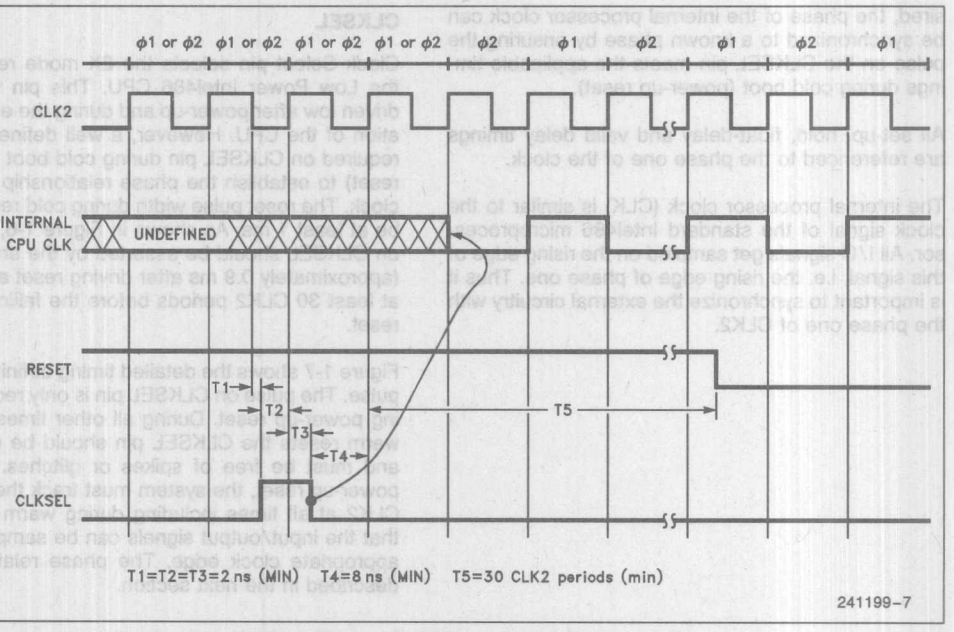


Figure 1-7. CLKSEL Timing Definition during Power-Up Reset

1.7 Architecture Overview

The Low Power Intel486 microprocessor is architecturally similar to the Intel486 CPU. Thus all bus cycles follow the same definition (for details see the Intel486 data sheet). The difference lies in the fact that the Low Power Intel486 CPU works with an external 2X clock input (CLK2). As shown in Figure 1-5, each of the internal processor clock (CLK) cycle is two CLK2 cycles wide. Thus a 25 MHz Low Power Intel486 microprocessor needs a 50 MHz clock input.

CLK2 provides the fundamental timing for the Low Power Intel486 CPU. It is divided by two internally to generate the internal processor clock (CLK) used for instruction execution. The internal clock is comprised of two phases, "phase one" and "phase two". Each CLK2 period is a phase of the internal clock. All Low Power Intel486 microprocessor inputs are sampled at the rising edge of phase 1. Each bus cycle is comprised of at least two bus states, T1 and T2. Each bus state in turn consists of two CLK2 cycles phase 1 and phase 2 of the bus state. The bus state diagram in Section 7.2.13 of the Intel486 CPU data sheet is valid for the Low Power Intel486 microprocessor.

NOTE:
The timing diagrams given in the Intel486 data sheet can be used for the Low Power Intel486 microprocessor. Read "CLK" signal as the internal clock of the CPU, with "CLK2" (the input clock of the Low Power Intel486 CPU) being twice the frequency of the internal processor clock as shown in Figure 1-5.

The following describes how the input signals are sampled and output signals are referenced with respect to the input clock (CLK2):

INPUT SIGNALS:

The Low Power Intel486 CPU samples all its **synchronous** input signals (i.e. RDY#, BRDY#, BS8#, BS16#, KEN#, EADS#, BOFF#, HOLD and AHOLD) at the rising edge of phase 1, as long as proper setup and hold times relative to that clock edge are met.

The Low Power Intel486 CPU samples all its **asynchronous** input signals (i.e. RESET, INTR, NMI, A20M# FLUSH#, IGNNE#) at every other rising edge of the system clock (Phase 1), as long as proper setup and hold times relative to that clock edge are met.

OUTPUT SIGNALS

The A.C. timing specifications for output signals (i.e. valid and float delay timings) are specified with respect to the rising edge of the Phase 1 of the system clock. This holds true for all output signals including ADS# and PCHK#.

2

1.8 Variable CPU Frequency

The Low Power Intel486 microprocessor allows the CPU frequency to change dynamically. As shown in Figures 1-8 and 1-9, the relationship between frequency and power consumption is approximately linear. Thus lowering the CPU frequency, reduces the power supply current (I_{CC}) consumed by the CPU.

The following must be satisified to change the CPU frequency:

1. Frequency can be changed at least 8 clocks after satisfying t4 (see Figure 1-7). The system can be started at a lower frequency and after satisfying the CLKSEL pulse specifications, it can be operated at the required speed.
2. The change in frequency should satisfy the minimum specification of "CLK2 high time" and "CLK2 low time". That is, at no time should the clock period go below the specified clock high and clock low times (see A.C. specifications for exact values).

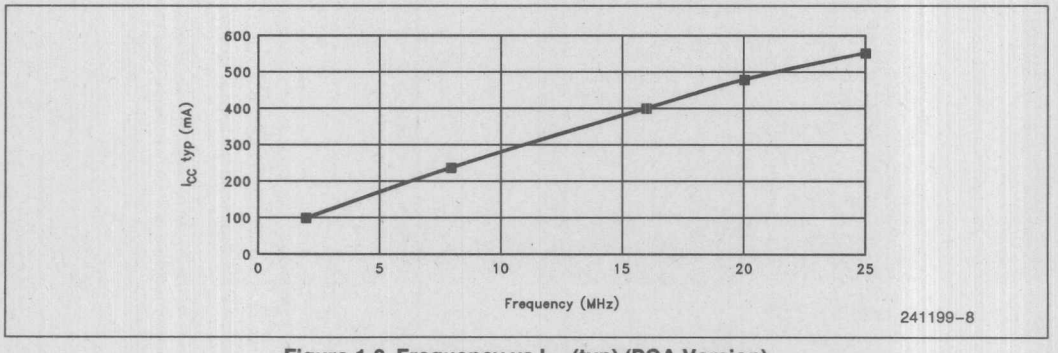


Figure 1-8. Frequency vs I_{CC}(typ) (PGA Version)

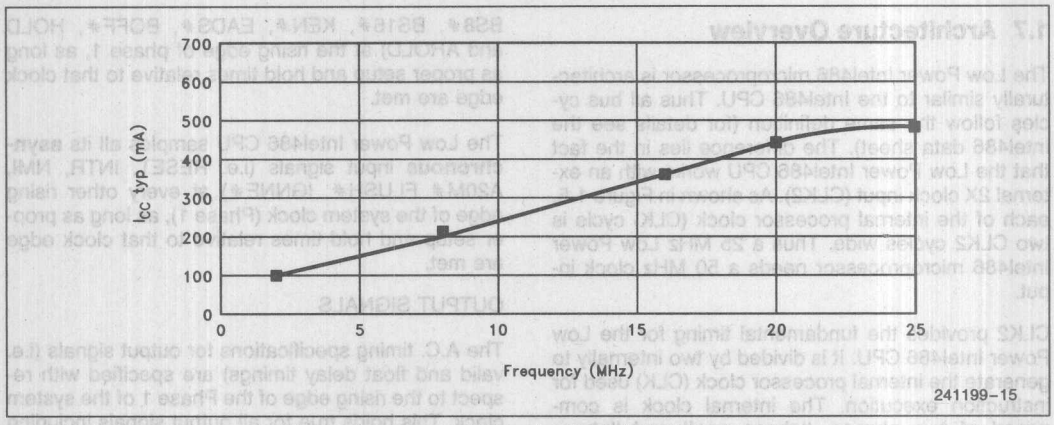


Figure 1-9. Frequency vs I_{CC}(typ) (PQFP Version)

2.0 D.C./A.C. SPECIFICATIONS

Table 2-1 provides the absolute maximum ratings. It is a stress rating only and functional operation at the maximums is not guaranteed. Functional operating conditions are given in Section 2.1 D.C. Specifications and 2.3 A.C. Specifications.

Table 2-1. Absolute Maximum Ratings

Case Temperature under Bias	−65°C to +110°C
Storage Temperature	−65°C to +150°C
Voltage on Any Pin with Respect to Ground	−0.5V to (V_{CC} +0.5V)
Supply Voltage with Respect to V_{SS}	−0.5V to +6.5V

2.1 D.C. Specifications

Table 2-2 provides the D.C. operating conditions for the Low Power Intel486 DX microprocessor.

Functional operating range: V_{CC} = 5V ±10%; T_{case} = 0°C to +85°C.

Table 2-2. Low Power Intel486 DX Microprocessor D.C. Parametric Values (PGA Version)

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current CLK2 = 50 MHz		700	mA	(Note 3)
I_{LI}	Input Leakage Current		±15	μA	(Note 4)
I_{IH}	Input Leakage Current		200	μA	(Note 5)
I_{IL}	Input Leakage Current		−400	μA	(Note 6)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		20	pF	F_c = 1 MHz(7)
C_O	I/O or Output Capacitance		20	pF	F_c = 1 MHz(7)
C_{CLK}	CLK Capacitance		20	pF	F_c = 1 MHz(7)

NOTES:
1. This parameter is measured at:
   Address, Data BEn 4.0 mA
   Definition, Control 5.0 mA
2. This parameter is measured at:
   Address, Data BEn −1.0 mA
   Definition, Control −0.9 mA
3. Typical supply current
   I_{CC} = 550 mA @CLK2 = 50 MHz
4. This parameter is for inputs without pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with pulldowns and V_{IH} = 2.4V.
6. This parameter is for inputs with pullups and V_{IL} = 0.45V.
7. Not 100% tested.

Table 2-3 provides the D.C. operating conditions for the Low Power Intel486 SX microprocessor (PGA Version) and the Intel487 SX Math CoProcessor installed in a low power system.

Functional Operating Range: V_{CC} = 5V ±10%; T_{case} = 0°C to +85°C.

Table 2-3. Low Power Intel486 SX Microprocessor D.C. Parametric Values (PGA Version)

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current CLK2 = 32 MHz = 40 MHz = 50 MHz		 525 600 700	 mA	(Note 3)
I_{CCF}	Power Supply Current with Intel486 SX CPU Tri-stated (floating) CLK2 = 32 MHz = 40 MHz = 50 MHz		 400 500 600	 mA	
I_{LI}	Input Leakage Current		±15	µA	(Note 4)
I_{IH}	Input Leakage Current		200	µA	(Note 5)
I_{IL}	Input Leakage Current		−400	µA	(Note 6)
I_{LO}	Output Leakage Current		±15	µA	
C_{IN}	Input Capacitance		20	pF	F_c = 1 MHz(7)
C_O	I/O or Output Capacitance		20	pF	F_c = 1 MHz(7)
C_{CLK}	CLK Capacitance		20	pF	F_c = 1 MHz(7)

NOTES:
1. This parameter is measured at:
   Address, Data BEn 4.0 mA
   Definition, Control 5.0 mA
2. This parameter is measured at:
   Address, Data BEn −1.0 mA
   Definition, Control −0.9 mA
3. Typical supply current
   I_{CC} = 400 @CLK2 = 32 MHz (Normal Operation)
   = 475 mA @CLK2 = 40 MHz
   = 500 mA @CLK2 = 50 MHz
   I_{CCF} = 325 mA @CLK2 = 32 MHz Intel486 CPU Tri-stated (Floating)
   = 400 mA @CLK2 = 40 MHz
   = 470 mA @CLK2 = 50 MHz
4. This parameter is for inputs without pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with pulldowns and V_{IH} = 2.4V.
6. This parameter is for inputs with pullups and V_{IL} = 0.45V.
7. Not 100% tested.

Table 2-4 provides the D.C. Operating Conditions for the Low Power Intel486 SX microprocessor (PQFP version) and the Intel487 SX Math CoProcessor installed in a low power system.

Functional Operating Range:

V_{CC} = 5V − 10%, +5%;
T_{CASE} = 0°C to +85°C

Table 2-4. Low Power Intel486™ SX Microprocessor D.C. Parametric Values (PQFP version)

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	24		V	(Note 2)
I_{CC}	Power Supply Current CLK2 = 32 MHz CLK2 = 40 MHz CLK2 = 50 MHz		 450 500 560	 mA	(Note 3)
I_{CCF}	Power Supply Current with Intel486 SX CPU in Power Down Mode		50	mA	(Note 7)
I_{LI}	Input Leakage Current		±15	μA	(Note 4)
I_{IH}	Input Leakage Current		200	μA	(Note 5)
I_{IL}	Input Leakage Current		−400	μA	(Note 6)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		10	pF	F_C = 1 MHz (Note 7)
C_O	I/O or Output Capacitance		10	pF	F_C = 1 MHz (Note 7)
C_{CLK}	CLK Capacitance		6	pF	F_C = 1 MHz (Note 7)

PRELIMINARY
SEE INTEL FOR DESIGN-IN
INFORMATION

NOTES:
1. This parameter is measured at:
   Address, Data, BEn 4.0 mA
   Definition, Control 5.0 mA
2. This parameter is measured at:
   Address, Data BEn −1.0 mA
   Definition, Control −0.9 mA
3. Typical supply current:
   I_{CC} 380 mA @ CLK2 = 32 MHz (Normal Operation)
   440 mA @ CLK2 = 40 MHz
   480 mA @ CLK2 = 50 MHz
4. This parameter is for inputs without pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with pulldowns and V_{IH} = 2.4V.
6. This parameter is for inputs with pullups and V_{IL} = 0.45V.
7. Not 100% tested.

2.2 Power Supply Current vs Frequency

Following is the power consumption of the Low Power Intel486 microprocessor or Intel487 Math CoProcessor installed in a low power system for different frequencies.

Table 2-5. Power Supply Current (I_{CC}) Values over Frequencies of Operation (PGA Version)

CLK2 Frequency (MHz)	Operating Frequency (MHz)	$I_{CC(max)}$ (mA)	$I_{CC(typ)}$ (mA)
4	2	150	100
16	8	325	235
32	16	525	400
40	20	600	475
50	25	700	550

Table 2-6. Power Supply Current (I_{CC}) Values over Frequencies of Operation (PQFP Version)

CLK2 Frequency (MHz)	Operating Frequency (MHz)	I_{CC} (max) (mA)	I_{CC} (typ) (mA)
4	2	150	100
16	8	250	210
32	16	450	380
40	20	500	440
50	25	560	480

2.3 A.C. Specifications

The following tables provide the A.C. specifications for the Low Power Intel486 microprocessors. They consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the rising edge of the phase 1 of the input system clock (CLK2), unless otherwise specified.

Table 2-7. Low Power Intel486 DX—25 MHz Microprocessor A.C. Characteristics

V_{CC} = 5V ±10%; T_{case} = 0°C to +85°C; C_L = 50 pF(2) unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	2	25	MHz		Half of CLK2 Frequency
t_1	CLK2 Period	20	250	ns	2.1	
t_2	CLK2 High Time	7		ns	2.1	At 2V
t_3	CLK2 Low Time	7		ns	2.1	At 0.8V
t_4	CLK2 Fall Time		2	ns	2.1	2V to 0.8V
t_5	CLK2 Rise Time		2	ns	2.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	22	ns	2.2	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		30	ns	2.2	After Clock Edge(1)
t_8	PCHK# Valid Delay	3	27	ns	2.2	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	27	ns	2.3	

Table 2-7. Low Power Intel486 DX—25 MHz Microprocessor A.C. Characteristics (Continued)
V_{CC} = 5V ±10%; T_{case} = 0°C to +85°C; C_L = 50 pF(2) unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_9	BLAST#, PLOCK# Float Delay		30	ns	2.2	After Clock Edge(1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	22	ns	2.2	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		30	ns	2.2	After Clock Edge(1)
t_{12}	EADS# Setup Time	9		ns	2.3	
t_{13}	EADS# Hold Time	4		ns	2.3	
t_{14}	KEN#, BS16#, BS8# Setup Time	9		ns	2.3	
t_{15}	KEN#, BS16#, BS8# Hold Time	4		ns	2.3	
t_{16}	RDY#, BRDY# Setup Time	9		ns	2.3	
t_{17}	RDY#, BRDY# Hold Time	4		ns	2.3	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	11		ns	2.3	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	4		ns	2.3	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	11		ns	2.3	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	4		ns	2.3	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	6		ns	2.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	4		ns	2.3	
	CLKSEL	See Figures 1-6 and 1-7 for details on this signal. Figure 1-7 shows minimum timings required for the proper operation of the CPU. The pulse on CLKSEL can be of any length as long as the minimums are satisfied and the transitions from low to high occurs at the clock edge shown.				

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF.

Table 2-8. Low Power Intel486™ SX—16 MHz Microprocessor/Intel487™ SX Math CoProcessor A.C. Characteristics

V_{CC} = 5V ±10%; T_{case} = 0°C to +85°C; C_L = 50 pF(2) unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	2	16	MHz		Half of CLK2 Frequency
t_1	CLK2 Period	31	250	ns	2.1	
t_2	CLK2 High Time	10		ns	2.1	At 2V
t_3	CLK2 Low Time	10		ns	2.1	At 0.8V
t_4	CLK2 Fall Time		4	ns	2.1	2V to 0.8V
t_5	CLK2 Rise Time		4	ns	2.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, **FERR#***, BREQ, HLDA Valid Delay	3	26	ns	2.2	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		42	ns	2.2	After Clock Edge(1)
t_8	PCHK# Valid Delay	3	35	ns	2.2	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	35	ns	2.2	
t_9	BLAST#, PLOCK# Float Delay		42	ns	2.2	After Clock Edge(1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	30	ns	2.2	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		42	ns	2.2	After Clock Edge(1)
t_{12}	EADS# Setup Time	12		ns	2.3	
t_{13}	EADS# Hold Time	4		ns	2.3	
t_{14}	KEN#, BS16#, BS8# Setup Time	12		ns	2.3	
t_{15}	KEN#, BS16#, BS8# Hold Time	4		ns	2.3	
t_{16}	RDY#, BRDY# Setup Time	12		ns	2.3	
t_{17}	RDY#, BRDY# Hold Time	4		ns	2.3	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	12		ns	2.3	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	4		ns	2.3	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, **IGNNE#*** Setup Time	14		ns	2.3	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, **IGNNE#*** Hold Time	4		ns	2.3	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	10		ns	2.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	4		ns	2.3	
	CLKSEL	See Figures 1-6 and 1-7 for details on this signal. Figure 1-7 shows minimum timings required for the proper operation of the CPU. The pulse on CLKSEL can be of any length as long as the minimums are satisfied and the transitions from low to high occurs at the clock edge shown.				

*Present only in the Intel487 SX Math CoProcessor

Table 2-9. Low Power Intel486™ SX—20 MHz Microprocessor/Intel487™ SX Math CoProcessor A.C. Characteristics

V_{CC} = 5V ±10%; T_{case} = 0°C to +85°C; C_L = 50 pF(2) unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	2	20	MHz		Half of CLK2 Frequency
t_1	CLK2 Period	25	250	ns	2.1	
t_2	CLK2 High Time	8.5		ns	2.1	At 2V
t_3	CLK2 Low Time	8.5		ns	2.1	At 0.8V
t_4	CLK2 Fall Time		3	ns	2.1	2V to 0.8V
t_5	CLK2 Rise Time		3	ns	2.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, **FERR#***, BREQ, HLDA Valid Delay	3	23	ns	2.2	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		37	ns	2.2	After Clock Edge(1)
t_8	PCHK# Valid Delay	3	28	ns	2.2	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	28	ns	2.2	
t_9	BLAST#, PLOCK# Float Delay		37	ns	2.2	After Clock Edge(1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	26	ns	2.2	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		37	ns	2.2	After Clock Edge(1)
t_{12}	EADS# Setup Time	10		ns	2.3	
t_{13}	EADS# Hold Time	4		ns	2.3	
t_{14}	KEN#, BS16#, BS8# Setup Time	10		ns	2.3	
t_{15}	KEN#, BS16#, BS8# Hold Time	4		ns	2.3	
t_{16}	RDY#, BRDY# Setup Time	10		ns	2.3	
t_{17}	RDY#, BRDY# Hold Time	4		ns	2.3	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	12		ns	2.3	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	4		ns	2.3	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, **IGNNE#*** Setup Time	12		ns	2.3	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, **IGNNE#*** Hold Time	4		ns	2.3	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	6		ns	2.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	4		ns	2.3	

*Present only in the Intel487 SX Math CoProcessor

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF.

Table 2-10. Low Power Intel486—25 MHz Microprocessor/Intel487™ SX Math CoProcessor A.C. Characteristics

V_{CC} = 5V ±10%; T_{case} = 0°C to +85°C; C_L = 50 pF(2) unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	2	25	MHz		Half of CLK2 Frequency
t_1	CLK2 Period	20	250	ns	2.1	
t_2	CLK2 High Time	7		ns	2.1	At 2V
t_3	CLK2 Low Time	7		ns	2.1	At 0.8V
t_4	CLK2 Fall Time		2	ns	2.1	2V to 0.8V
t_5	CLK2 Rise Time		2	ns	2.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, **FERR#***, BREQ, HLDA Valid Delay	3	19	ns	2.2	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		28	ns	2.2	After Clock Edge(1)
t_8	PCHK# Valid Delay	3	24	ns	2.2	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	24	ns	2.2	
t_9	BLAST#, PLOCK# Float Delay		28	ns	2.2	After Clock Edge(1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	20	ns	2.2	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		28	ns	2.2	After Clock Edge(1)
t_{12}	EADS# Setup Time	9		ns	2.3	
t_{13}	EADS# Hold Time	4		ns	2.3	
t_{14}	KEN#, BS16#, BS8# Setup Time	9		ns	2.3	
t_{15}	KEN#, BS16#, BS8# Hold Time	4		ns	2.3	
t_{16}	RDY#, BRDY# Setup Time	9		ns	2.3	
t_{17}	RDY#, BRDY# Hold Time	4		ns	2.3	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	11		ns	2.3	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	4		ns	2.3	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, **IGNNE#*** Setup Time	11		ns	2.3	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, **IGNNE#*** Hold Time	4		ns	2.3	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	6		ns	2.3	

*Present only in the Intel487 SX Math CoProcessor

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF.

Table 2-10. Low Power Intel486—25 MHz Microprocessor/Intel487™ SX Math CoProcessor A.C. Characteristics

V_{CC} = 5V ±10%; T_{case} = 0°C to +85°C; C_L = 50 pF(2) unless otherwise specified (Continued)

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	4		ns	2.3	
	CLKSEL	See Figures 1-6 and 1-7 for details on this signal. Figure 1-7 shows minimum timings required for the proper operation of the CPU. The pulse on CLKSEL can be of any length as long as the minimums are satisfied and the transitions from low to high occurs at the clock edge shown.				

*Present only in the Intel487 SX Math CoProcessor

NOTES:
1. Not 100% tested, guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF.

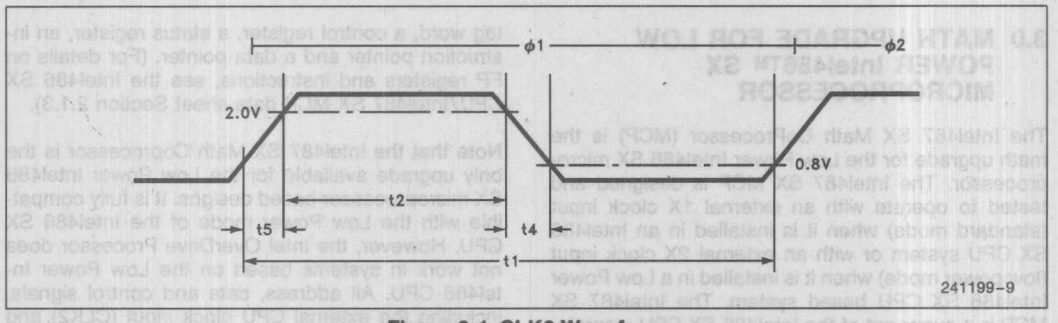


Figure 2-1. CLK2 Waveform

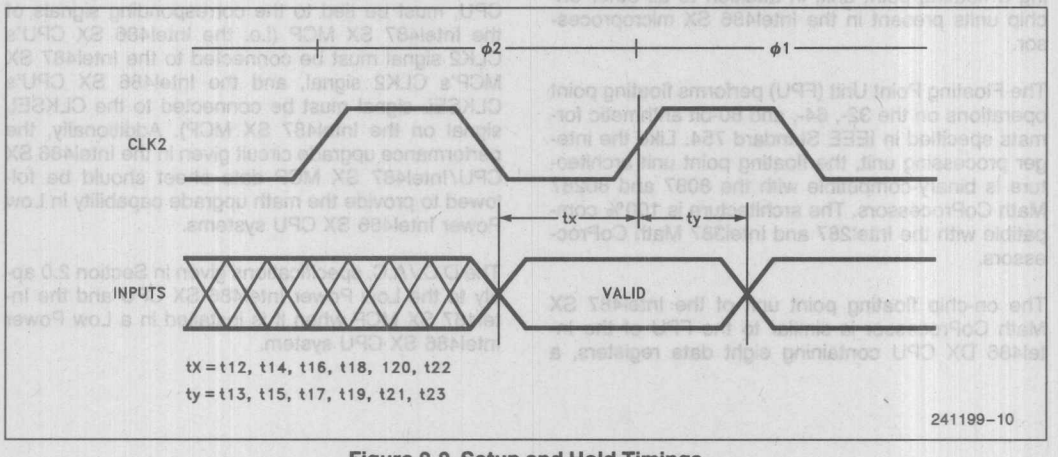


Figure 2-2. Setup and Hold Timings

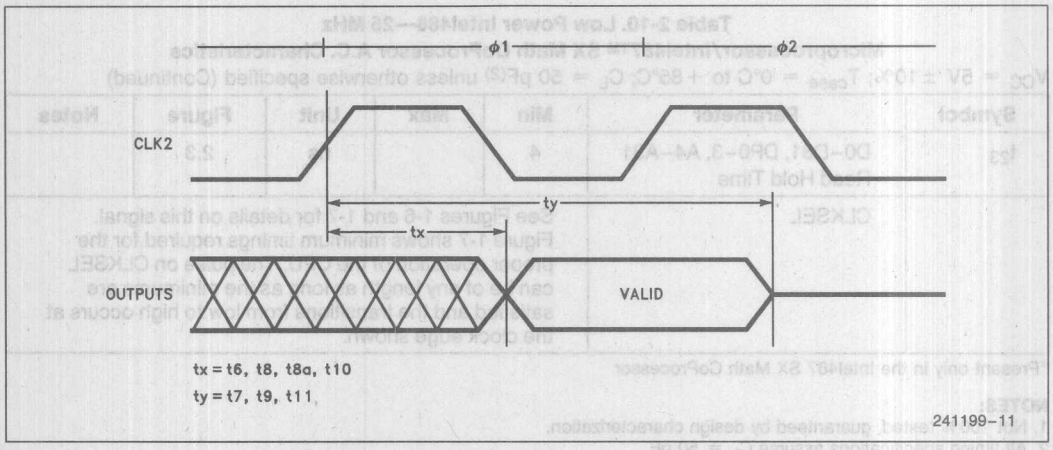


Figure 2-3. Valid and Float Delay Timings

3.0 MATH UPGRADE FOR LOW POWER Intel486™ SX MICROPROCESSOR

The Intel487 SX Math CoProcessor (MCP) is the math upgrade for the Low Power Intel486 SX microprocessor. The Intel487 SX MCP is designed and tested to operate with an external 1X clock input (standard mode) when it is installed in an Intel486 SX CPU system or with an external 2X clock input (low power mode) when it is installed in a Low Power Intel486 SX CPU based system. The Intel487 SX MCP is a super-set of the Intel486 SX CPU, containing a floating point unit, in addition to all other on-chip units present in the Intel486 SX microprocessor.

The Floating Point Unit (FPU) performs floating point operations on the 32-, 64-, and 80-bit arithmetic formats specified in IEEE Standard 754. Like the integer processing unit, the floating point unit architecture is binary-compatible with the 8087 and 80287 Math CoProcessors. The architecture is 100% compatible with the Intel287 and Intel387 Math CoProcessors.

The on-chip floating point unit of the Intel487 SX Math CoProcessor is similar to the FPU of the Intel486 DX CPU containing eight data registers, a tag word, a control register, a status register, an instruction pointer and a data pointer. (For details on FP registers and instructions, see the Intel486 SX CPU/Intel487 SX MCP data sheet Section 2.1.3).

Note that the Intel487 SX Math Coprocessor is the only upgrade available for the Low Power Intel486 SX microprocessor based designs. It is fully compatible with the Low Power mode of the Intel486 SX CPU. However, the Intel OverDrive Processor does not work in systems based on the Low Power Intel486 CPU. All address, data and control signals, including the external CPU clock input (CLK2) and the CLKSEL signal of the Low Power Intel486 SX CPU, must be tied to the corresponding signals of the Intel487 SX MCP (i.e. the Intel486 SX CPU's CLK2 signal must be connected to the Intel487 SX MCP's CLK2 signal, and the Intel486 SX CPU's CLKSEL signal must be connected to the CLKSEL signal on the Intel487 SX MCP). Additionally, the performance upgrade circuit given in the Intel486 SX CPU/Intel487 SX MCP data sheet should be followed to provide the math upgrade capability in Low Power Intel486 SX CPU systems.

The D.C./A.C. specifications given in Section 2.0 apply to the Low Power Intel486 SX CPU and the Intel487 SX MCP when it is installed in a Low Power Intel486 SX CPU system.

3.1 Pinout

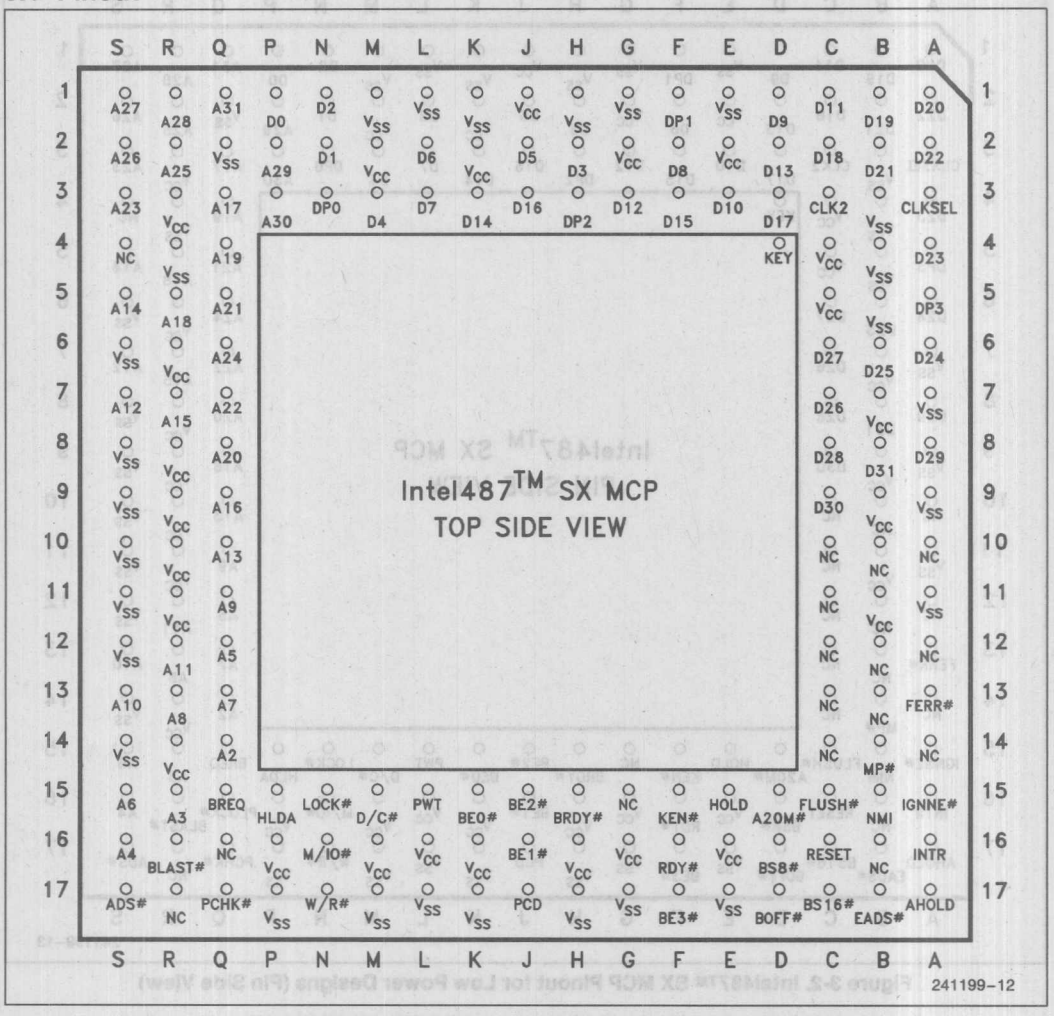


Figure 3-1. Intel487™ SX MCP Pinout for Low Power Designs (Top Side View)

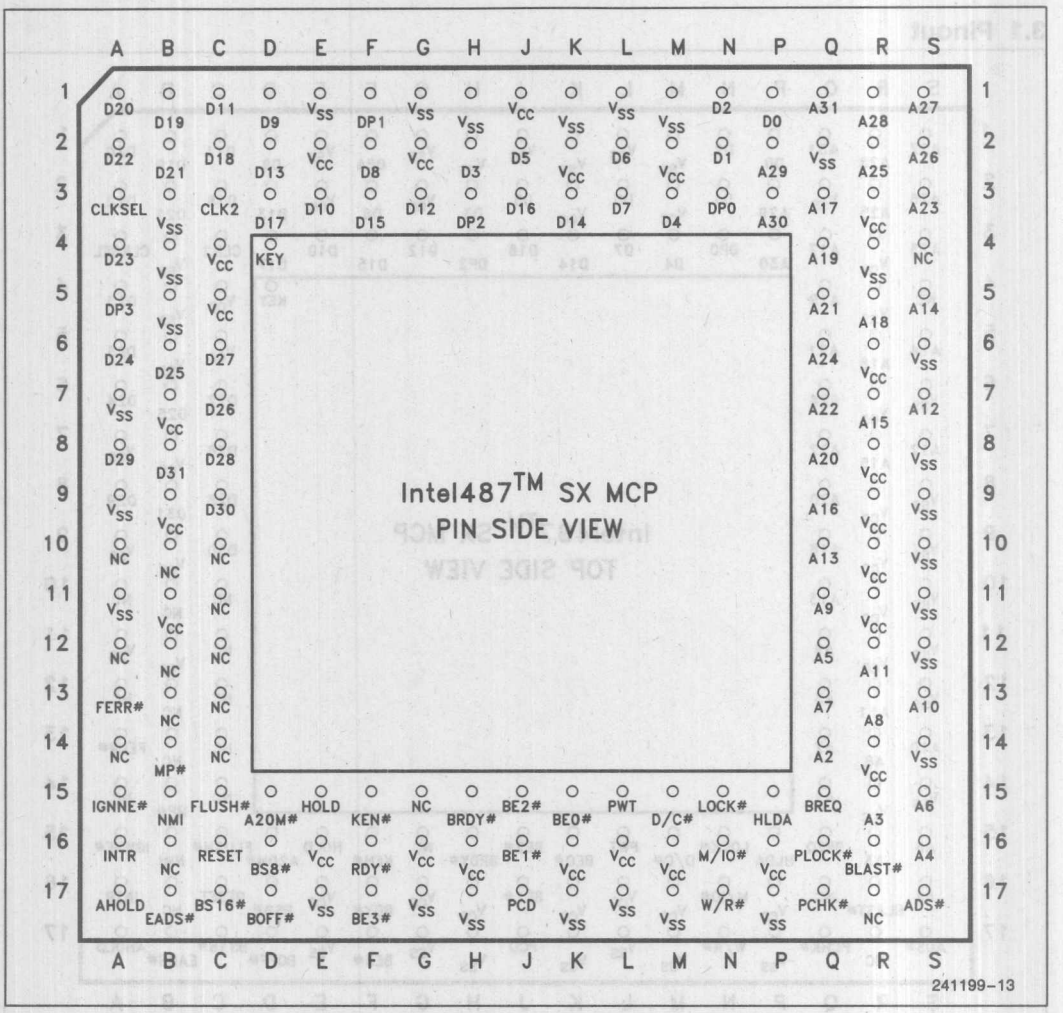


Figure 3-2. Intel487™ SX MCP Pinout for Low Power Designs (Pin Side View)

3.2. Pin Reference of Intel487™ SX Math CoProcessor

Table 3-1. Pin Cross Reference by Pin Name

Address		Data		Control		N/C	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	A10	B7	A7
A_3	R15	D_1	N2	ADS#	S17	A12	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	A14	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	B10	C4	B3
A_6	S15	D_4	M3	BE1#	J16	B12	C5	B4
A_7	Q13	D_5	J2	BE2#	J15	B13	E2	B5
A_8	R13	D_6	L2	BE3#	F17	B16	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16	C10	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17	C11	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15	C12	H16	G17
A_{12}	S7	D_{10}	E3	BREQ	Q15	C13	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16	C14	K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17	G15	K16	K1
A_{15}	R7	D_{13}	D2	CLK2	C3	R17	L16	K17
A_{16}	Q9	D_{14}	K3	CLKSEL	A3	S4	M2	L1
A_{17}	Q3	D_{15}	F3	D/C#	M15		M16	L17
A_{18}	R5	D_{16}	J3	DP0	N3		P16	M1
A_{19}	Q4	D_{17}	D3	DP1	F1		R3	M17
A_{20}	Q8	D_{18}	C2	DP2	H3		R6	P17
A_{21}	Q5	D_{19}	B1	DP3	A5		R8	Q2
A_{22}	Q7	D_{20}	A1	EADS#	B17		R9	R4
A_{23}	S3	D_{21}	B2	FERR#	A13		R10	S6
A_{24}	Q6	D_{22}	A2	FLUSH#	C15		R11	S8
A_{25}	R2	D_{23}	A4	HLDA	P15		R14	S9
A_{26}	S2	D_{24}	A6	HOLD	E15			S10
A_{27}	S1	D_{25}	B6	IGNNE#	A15			S11
A_{28}	R1	D_{26}	C7	INTR	A16			S12
A_{29}	P2	D_{27}	C6	KEN#	F15			S14
A_{30}	P3	D_{28}	C8	LOCK#	N15			
A_{31}	Q1	D_{29}	A8	M/IO#	N16			
		D_{30}	C9	MP#	B14			
		D_{31}	B8	NMI	B15			
				PCD	J17			
				PCHK#	Q17			
				PWT	L15			
				PLOCK#	Q16			
				RDY#	F16			
				RESET	C16			
				W/R#	N17			

3.3. Intel487™ SX Math CoProcessor Pin Description

The Intel487 SX Math CoProcessor consists of all the Intel486 SX microprocessor signals with the following additional pins.

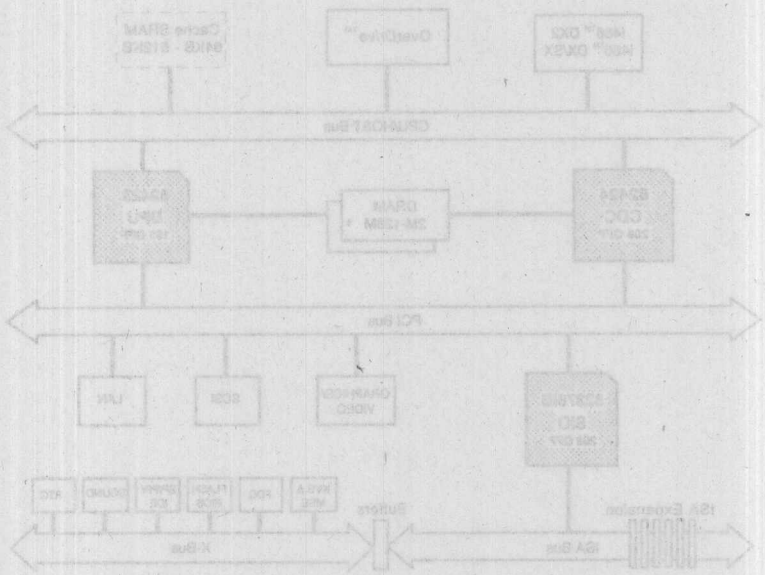
NUMERIC ERROR REPORTING		
FERR#	O	The **Floating point error** pin is driven active when a floating point error occurs. FERR# is similar to the ERROR# pin on the i387 Math CoProcessor. FERR# is included for compatibility with systems using DOS type floating point error reporting. FERR# is active LOW, and is not floated CONT1g bus hold.
IGNNE#	I	When the **ignore numeric error** pin is asserted, the Low Power Intel487 SX Math CoProcessor will ignore a numeric error and continue executing non-control floating point instructions. When IGNNE# is deasserted the Low Power Intel487 SX Math CoProcessor will freeze on a non-control floating point instruction, if a previous floating point instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set. IGNNE# is active LOW and is provided with a small internal pullup resistor. IGNNE# is asynchronous but setup and hold times t_{20} and t_{21} must be met to insure recognition on any specific clock.
Intel487™ SX MATH COPROCESSOR INTERFACE		
MP#	O	The **math present** pin is used to signal the Intel486 SX microprocessor to float its outputs and get-off the bus. This pin can be used to check the presence of the Math CoProcessor in the two socket math upgrade circuit. It is active low and is never floated. MP# is driven low at power-up and remains active for the entire duration of the Intel487™ SX Math CoProcessor operation.
KEY PIN		
KEY		The **KEY** pin is an electrically non-functional pin which is used to insure the correct Intel487 SX Math CoProcessor orientation in a 169-pin socket. KEY pin is located at "D4" and is the 169th pin of the Intel487 SX MCP.

4.0 REVISION HISTORY

Revision-002 of the Low Power Intel486 SX CPU/Intel487 SX MCP data book contains many updates and improvements to the original version. A revision summary of major changes is listed below:

The sections significantly revised since version -001 are:

Cover Page	Added Low Power Intel486 SX CPU PQFP package information.
Section 1.1	Added Figure 1-5. Low Power Intel486 SX CPU PQFP Pinout.
Section 1.4	Added Low Power Intel486 SX CPU PQFP package to Pin Cross Reference Table.
Section 1.5	Added Performance Upgrade Support section of Low Power Intel486 SX CPU PQFP package.
	Added Table 1-5. Test Pins.
	Added Table 1-6 Component ID and Revision ID.
	Added Test Access Port section of Low Power Intel486 SX CPU PQFP package.
Table 1-2	Noted IGNNE# pin is present on the Intel486 DX CPU and Intel487 SX MCP only.
	Added UP# pin and descriptions.
Section 1.7	Added Figure 1-9. Frequency vs I_{CC} (typ) (PQFP Version).
Section 2.1	Added Table 2-4. Intel486 SX Microprocessor D.C. Parametric Values (for PQFP package).
Section 2.2	Added Table 2-6 Power Supply Current ($_{CC}$) Values over Frequencies of Operation (PQFP Version).



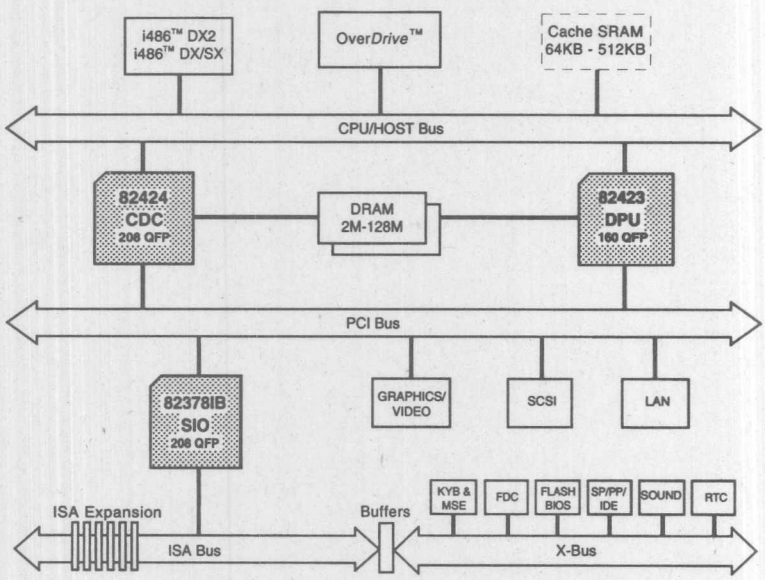
82420 PCIset

Intel's 82420 PCIset enables workstation level of performance for Intel486™ CPU desktop systems. The Peripheral Component Interconnect Bus (PCI) is driving a new architecture for PC's—eliminating the I/O bottleneck of standard expansion busses. PCI provides a glueless interface for high performance peripherals such as graphics, SCSI, LAN and video to be placed onto a fast local bus. By utilizing this technology and incorporating read/write bursts along with write buffers into the 82420 PCIset, a new level of performance is now possible for today's Intel486 CPU desktop systems.

The Intel 82420 PCIset is comprised of three components: the 82424 Cache DRAM Controller (CDC), the 82423 Data Path Unit (DPU), and the 82378 System I/O (SIO). The CDC and DPU provide the core system architecture while the SIO is a PCI master/slave agent which bridges the core architecture to the ISA standard expansion bus. Intel also offers two components, the 82374EB (ESC) and 82375EB (PCEB), that work in conjunction to bridge the PCI bus to the EISA expansion bus. Refer to the ESC and PCEB data sheets for information regarding the EISA bridge components.

The chip set supports the Intel486 CPU family as well as the write-back caching capability of Intel's future OverDrive™ processor for the Intel486 DX2 CPU. The high performance memory subsystem supports concurrent operation between PCI bus masters while the CPU accesses memory. An integrated second level cache can be programmed for write-through or write-back operation.

82420 PCIset



290467-1

Intel486 SX, Intel486 DX, Intel486 DX2, and OverDrive are trademarks of Intel Corporation.

October 1993
Order Number: 290467-003

Product Highlights

82424—Cache DRAM Controller (CDC)

- Concurrent Linefill during Copyback Cycles
- Supports Intel486 CPU Family and OverDrive Processors
- Supports Future OverDrive Upgrade Processor in Write-Back Cache Mode
- 64K–512K Level 2 Cache Support
- Level 2 Cache Configurable as Write-Back or Write-Through
- 208-Pin QFP Package

82423—Data Path Unit (DPU)

- Highly Integrated
- Four Dword Write Buffers
- Zero Wait States for CPU Write Cycles
- PCI Burst Write Capability
- 160-Pin QFP Package

82378—System I/O Component (SIO)

- Supports Fast DMA Type A, B, or F Cycles
- Supports DMA Scatter/Gather
- Arbitration Logic for Four PCI Masters
- Reusable across Multiple Platforms
- Directly Drives Six External ISA Slots
- Integrates Many of Today's Common I/O Functions
- 208-Pin QFP Package

Product Description

The 82424 Cache DRAM Controller (CDC) is a single-chip bridge from the CPU to the PCI bus. It provides the integrated functionality of a second level cache controller, a DRAM controller, and a PCI bus controller. It also features an optimized memory subsystem. The CDC is a dual ported device with one port as the host port and the other as the PCI port.

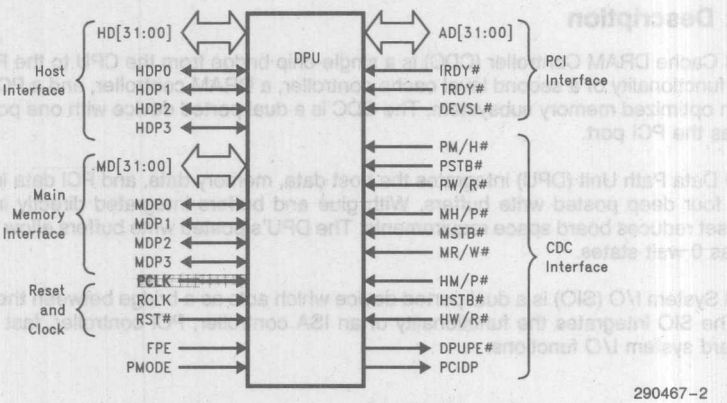
The 82423 Data Path Unit (DPU) integrates the host data, memory data, and PCI data interface, DPU control/parity and four deep posted write buffers. With glue and buffers integrated directly into the DPU, the Intel 82420 PCIset reduces board space requirements. The DPU's posted write buffers allow CPU write cycles to be executed as 0 wait states.

The 82378 System I/O (SIO) is a dual ported device which acts as a bridge between the PCI and standard ISA I/O bus. The SIO integrates the functionality of an ISA controller, PCI controller, fast 32-bit DMA controller, and standard system I/O functions.

82423
DATA PATH UNIT (DPU)

- **A 32-bit High Performance Host/PCI/ Memory Data Path**
- **Operates Synchronous to the CPU and PCI Clocks**
- **Dual-Port Architecture Allows Concurrent Operations on the Host and PCI Buses**
- **Burst Read of Memory from the Host and PCI Buses**
- **Host-to-Memory and Host-to-PCI Post Buffers Permit Zero Wait State Write Performance**
- **Byte Parity Support for the Host and Memory Buses**
  - **— Optional Parity Generation for Host-to-Memory Transfers**
  - **— Optional Parity Checking for the Secondary Cache Residing on the Host Data Bus**
  - **— Parity Checking for Host and PCI Memory Reads**
  - **— Parity Generation for PCI-to-Memory Writes**
- **Force Bad Parity to Memory Capability for Diagnostic Purposes**

The 82423 Data Path Unit (DPU) provides the 32-bit data path connections between the Host (CPU/cache), main memory, and the Peripheral Component Interconnect (PCI) Bus. The dual-port architecture allows concurrent operations on the Host and PCI Buses. Two 4-Dword deep Post buffers permit Host posting of data to main memory and the PCI Bus. The DPU supports byte parity for the Host and main memory buses. The DPU is intended to be used with the 82424 Cache DRAM Controller (CDC). During bus operations between the Host, main memory, and PCI, the CDC provides the address paths and bus controls. The CDC also controls the data flow through the DPU. Together, these two chips provide a full function dual-port data path connection to main memory and forms a Host/PCI bridge.



290467-2

IMPORTANT—READ THIS SECTION BEFORE READING THE REST OF THE DATA SHEET.

This data sheet describes the 82423TX and 82423ZX components. All normal text describes the functionality for both components. All features that exist on the 82423ZX are shaded as shown below.

This is an example of what the shaded sections that apply only to the 82423ZX component look like.

82424
CACHE AND DRAM CONTROLLER (CDC)

- Supports 25/33/*50* MHz Intel486™ SX, Intel487™ SX, Intel486 DX, Intel486 DX2, OverDrive™ for Intel486 and OverDrive for DX2 Processors
- Fully Synchronous, 25/33 MHz PCI Bus Capable of Supporting Bus Masters
- Supports OverDrive Upgrade Socket, Including OverDrive for DX2 in Write-Back Mode
- Programmable Attribute Map for First 1 MByte of Main Memory
- Posted Write Buffers for Improved Performance
- Integrated DRAM Controller
  - — 2 to 160 MByte Main Memory using 70 ns Fast Page Mode SIMM Memory
  - — Decoupled Refresh Cycles to Reduce DRAM Access Latency
  - — Burst Mode PCI Accesses to DRAM Supported at the Rate of x-3-3-3-3-3
- Integrated Cache Controller
  - — Write-Through and Write-Back Cache Options
  - — 64 KB, 128 KB, 256 KB and 512 KB Cache Sizes using Standards SRAMs
  - — Burst Line Fill of 2-1-1-1 from Secondary Cache at 25 and 33 MHz *and 3-1-1-1 at 50 MHz*
  - — Zero Wait State Write to L2 Cache for a Cacche Write Hit
  - — Main Memory Posting at Zero Wait States, Enabling Optimum Write-Through Cache Performance
  - — Concurrent Cache Line Replacement from Secondary Cache in Write-Back Mode
- PCI Bridge
  - — Translates CPU Cycles into PCI Bus Cycles
  - — Translates Back-to-Back Sequential Memory Write Cycles into PCI Burst Cycles
  - — Separate PCI-to-Main Memory Port Allows Concurrent/Independent CPU and PCI Bus Operations
  - — Integrated Snoop Filter

2

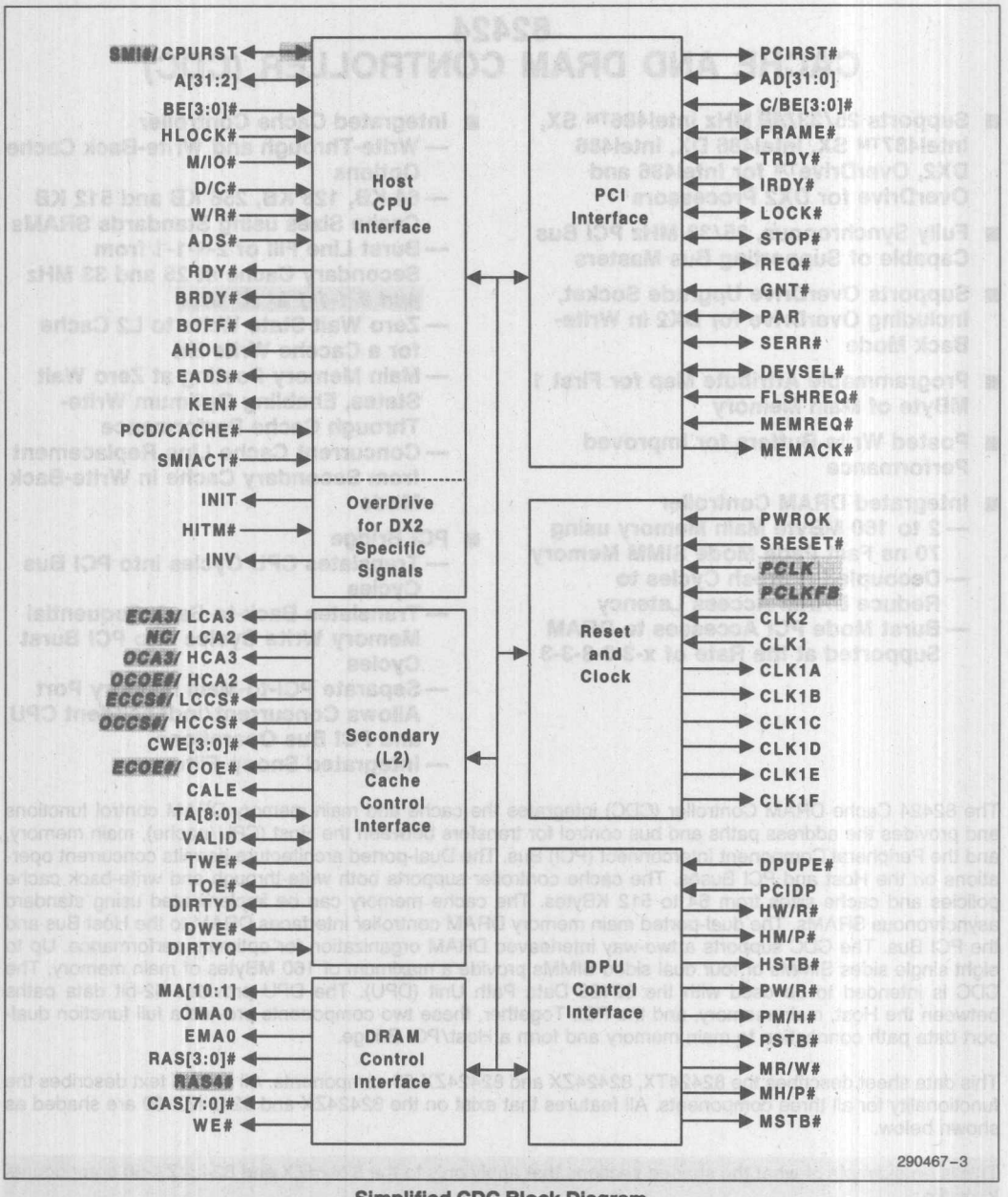
The 82424 Cache DRAM Controller (CDC) integrates the cache and main memory DRAM control functions and provides the address paths and bus control for transfers between the Host (CPU/cache), main memory, and the Peripheral Component Interconnect (PCI) Bus. The Dual-ported architecture permits concurrent operations on the Host and PCI Buses. The cache controller supports both write-through and write-back cache policies and cache sizes from 64 to 512 KBytes. The cache memory can be implemented using standard asynchronous SRAMs. The dual-ported main memory DRAM controller interfaces DRAM to the Host Bus and the PCI Bus. The CDC supports a two-way interleaved DRAM organization for optimum performance. Up to eight single sides SIMMs or four dual sided SIMMs provide a maximum of 160 MBytes of main memory. The CDC is intended to be used with the 82423 Data Path Unit (DPU). The DPU provides 32-bit data paths between the Host, main memory, and the PCI. Together, these two components provide a full function dual-port data path connection to main memory and form a Host/PCI Bridge.

This data sheet describes the 82424TX, 82424ZX and 82424ZX-50 components. All normal text describes the functionality for all three components. All features that exist on the 82424ZX and 82424ZX-50 are shaded as shown below.

This is an example of what the shaded sections that apply only to the 82424ZX and 82424ZX-50 components look like.

All features that exist only on the 82424ZX-50 are shaded as shown below.

This is an example of what the shaded sections that apply only to the 82424ZX-50 component look like.



Simplified CDC Block Diagram

intel

82378
SYSTEM I/O (SIO)

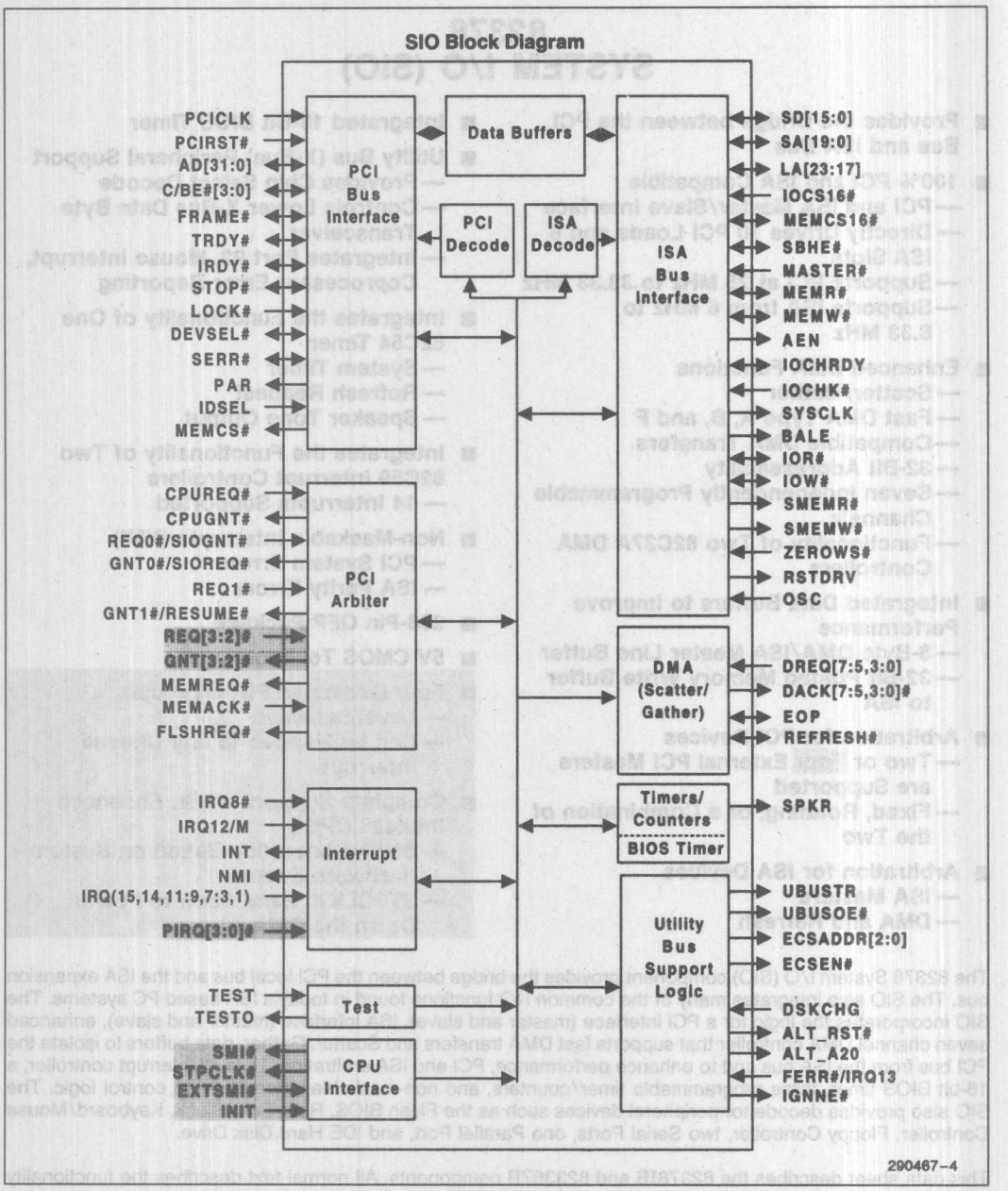
- Provides the Bridge between the PCI Bus and ISA Bus
- 100% PCI and ISA Compatible
  - PCI and ISA Master/Slave Interface
  - Directly Drives 10 PCI Loads and 6 ISA Slots
  - Supports PCI at 25 MHz to 33.33 MHz
  - Supports ISA from 6 MHz to 8.33 MHz
- Enhanced DMA Functions
  - Scatter/Gather
  - Fast DMA Type A, B, and F
  - Compatible DMA Transfers
  - 32-Bit Addressability
  - Seven Independently Programmable Channels
  - Functionality of Two 82C37A DMA Controllers
- Integrated Data Buffers to Improve Performance
  - 8-Byte DMA/ISA Master Line Buffer
  - 32-Bit Posted Memory Write Buffer to ISA
- Arbitration for PCI Devices
  - Two or Four External PCI Masters are Supported
  - Fixed, Rotating, or a Combination of the Two
- Arbitration for ISA Devices
  - ISA Masters
  - DMA and Refresh
- Integrated 16-Bit BIOS Timer
- Utility Bus (X-Bus) Peripheral Support
  - Provides Chip Select Decode
  - Controls Lower X-Bus Data Byte Transceiver
  - Integrates Port 92, Mouse Interrupt, Coprocessor Error Reporting
- Integrates the Functionality of One 82C54 Timer
  - System Timer
  - Refresh Request
  - Speaker Tone Output
- Integrates the Functionality of Two 82C59 Interrupt Controllers
  - 14 Interrupts Supported
- Non-Maskable Interrupts (NMI)
  - PCI System Errors
  - ISA Parity Errors
- 208-Pin QFP Package
- 5V CMOS Technology
- Four Dedicated PCI Interrupts
  - Level Sensitive
  - Can be Mapped to any Unused Interrupt
- Complete Support for SL Enhanced Intel486 CPU's
  - SMI# Generation Based on System Hardware Events
  - STPCLK# Generation to Power Down the CPU

2

The 82378 System I/O (SIO) component provides the bridge between the PCI local bus and the ISA expansion bus. The SIO also integrates many of the common I/O functions found in today's ISA based PC systems. The SIO incorporates the logic for a PCI interface (master and slave), ISA interface (master and slave), enhanced seven channel DMA controller that supports fast DMA transfers and Scatter/Gather, data buffers to isolate the PCI bus from the ISA bus and to enhance performance, PCI and ISA arbitration, 14 level interrupt controller, a 16-bit BIOS timer, three programmable timer/counters, and non-maskable-interrupt (NMI) control logic. The SIO also provides decode for peripheral devices such as the Flash BIOS, Real Time Clock, Keyboard/Mouse Controller, Floppy Controller, two Serial Ports, one Parallel Port, and IDE Hard Disk Drive.

This data sheet describes the 82378IB and 82338ZB components. All normal text describes the functionality for both components. All features that exist on the 82378ZB are shaded as shown below.

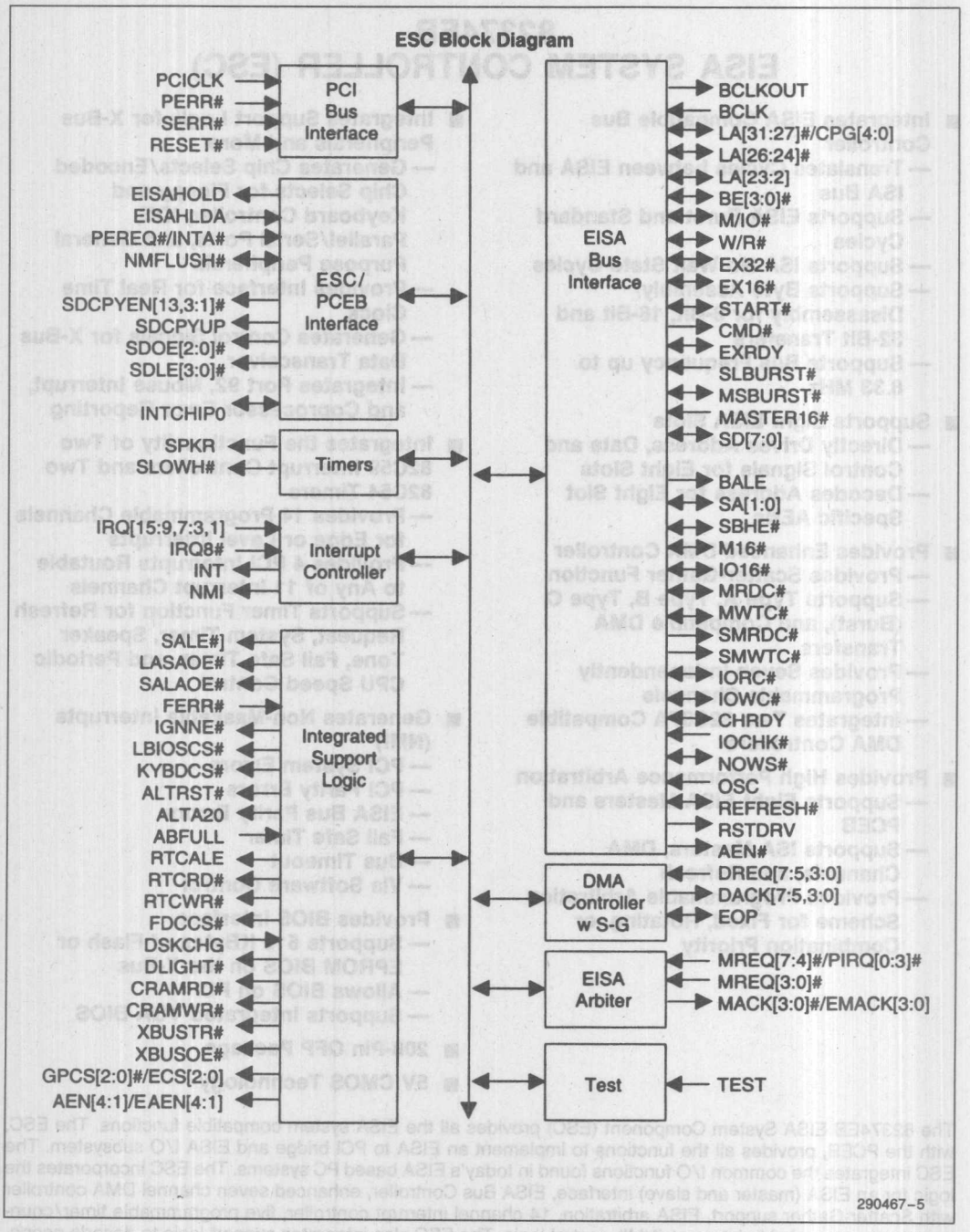
This is an example of what the shaded sections that apply only to the 82378ZB component look like.


SIO Block Diagram
PCICLK
PCIRST#
AD[31:0]
C/BE#[3:0]
FRAME#
TRDY#
IRDY#
STOP#
LOCK#
DEVSEL#
SERR#
PAR
IDSEL
MEMCS#
PCI Bus Interface
Data Buffers
PCI Decode
ISA Decode
ISA Bus Interface
SD[15:0]
SA[19:0]
LA[23:17]
IOCS16#
MEMCS16#
SBHE#
MASTER#
MEMR#
MEMW#
AEN
IOCHRDY
IOCHK#
SYSCLK
BALE
IOR#
IOW#
SMEMR#
SMEMW#
ZEROWS#
RSTDRV
OSC
CPUREQ#
CPUGNT#
REQ0#/SIOGNT#
GNT0#/SIOREQ#
REQ1#
GNT1#/RESUME#
REQ[3:2]#
GNT[3:2]#
MEMREQ#
MEMACK#
FLSHREQ#
PCI Arbiter
DMA (Scatter/ Gather)
DREQ[7:5,3:0]
DACK[7:5,3:0]#
EOP
REFRESH#
IRQ8#
IRQ12/M
INT
NMI
IRQ(15,14,11:9,7:3,1)
PIRQ[3:0]#
Interrupt
Timers/ Counters
BIOS Timer
SPKR
Utility Bus Support Logic
UBUSTR
UBUSOE#
ECSADDR[2:0]
ECSEN#
TEST
TESTO
Test
DSKCHG
ALT_RST#
ALT_A20
FERR#/IRQ13
IGNNE#
SMI#
STPCLK#
EXTSMI#
INIT
CPU Interface
290467-4

82374EB
EISA SYSTEM CONTROLLER (ESC)

- **Integrates EISA Compatible Bus Controller**
  - — Translates Cycles between EISA and ISA Bus
  - — Supports EISA Burst and Standard Cycles
  - — Supports ISA No Wait State Cycles
  - — Supports Byte Assembly/ Disassembly for 8-Bit, 16-Bit and 32-Bit Transfers
  - — Supports Bus Frequency up to 8.33 MHz
- **Supports Eight EISA Slots**
  - — Directly Drives Address, Data and Control Signals for Eight Slots
  - — Decodes Address for Eight Slot Specific AENs
- **Provides Enhanced DMA Controller**
  - — Provides Scatter-Gather Function
  - — Supports Type A, Type B, Type C (Burst), and Compatible DMA Transfers
  - — Provides Seven Independently Programmable Channels
  - — Integrates Two 82C37A Compatible DMA Controllers
- **Provides High Performance Arbitration**
  - — Supports Eight EISA Masters and PCEB
  - — Supports ISA Masters, DMA Channels, and Refresh
  - — Provides Programmable Arbitration Scheme for Fixed, Rotating, or Combination Priority
- **Integrates Support Logic for X-Bus Peripherals and More**
  - — Generates Chip Selects/Encoded Chip Selects for Floppy and Keyboard Controller, IDE, Parallel/Serial Ports, and General Purpose Peripherals
  - — Provides Interface for Real Time Clock
  - — Generates Control Signals for X-Bus Data Transceiver
  - — Integrates Port 92, Mouse Interrupt, and Coprocessor Error Reporting
- **Integrates the Functionality of Two 82C59 Interrupt Controllers and Two 82C54 Timers**
  - — Provides 14 Programmable Channels for Edge or Level Interrupts
  - — Provides 4 PCI Interrupts Routable to Any of 11 Interrupt Channels
  - — Supports Timer Function for Refresh Request, System Timer, Speaker Tone, Fail Safe Timer, and Periodic CPU Speed Control
- **Generates Non-Maskable Interrupts (NMI)**
  - — PCI System Errors
  - — PCI Parity Errors
  - — EISA Bus Parity Errors
  - — Fail Safe Timer
  - — Bus Timeout
  - — Via Software Control
- **Provides BIOS Interface**
  - — Supports 512 KBytes of Flash or EPROM BIOS on the X-Bus
  - — Allows BIOS on PCI
  - — Supports Integrated VGA BIOS
- **208-Pin QFP Package**
- **5V CMOS Technology**

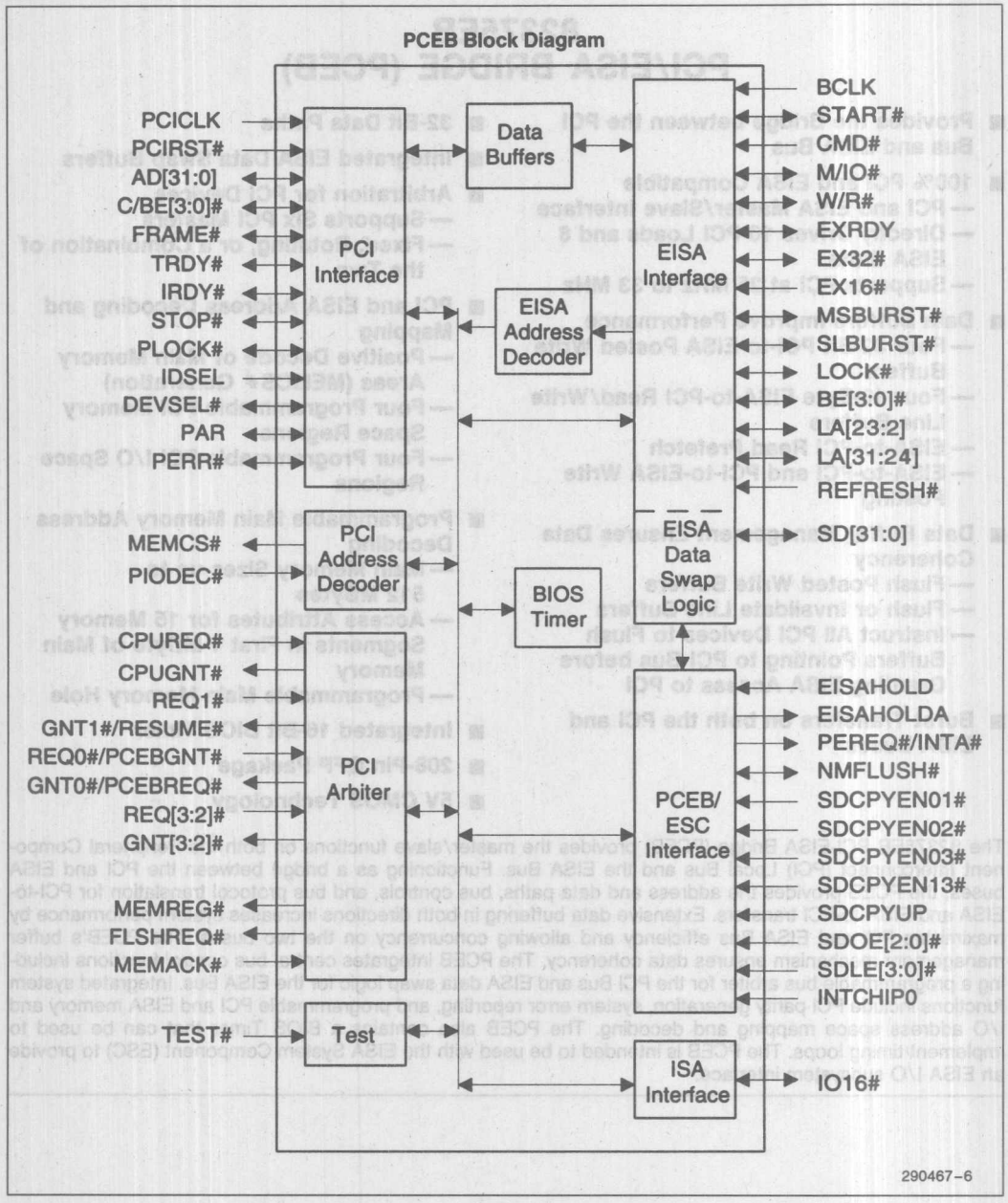
The 82374EB EISA System Component (ESC) provides all the EISA system compatible functions. The ESC, with the PCEB, provides all the functions to implement an EISA to PCI bridge and EISA I/O subsystem. The ESC integrates the common I/O functions found in today's EISA based PC systems. The ESC incorporates the logic for an EISA (master and slave) interface, EISA Bus Controller, enhanced seven channel DMA controller with Scatter-Gather support, EISA arbitration, 14 channel interrupt controller, five programmable timer/counters, and non-maskable interrupt (NMI) control logic. The ESC also integrates support logic to decode peripheral devices such as the Flash BIOS, Real Time Clock, Keyboard/Mouse Controller, Floppy Controller, two Serial Ports, one Parallel Port, and IDE Hard Disk Drive.


ESC Block Diagram
PCICLK
PERR#
SERR#
RESET#
PCI Bus Interface
EISAHOLD
EISAHLDA
PEREQ#/INTA#
NMFLUSH#
SDCPYEN[13,3:1]#
SDCPYUP
SDOE[2:0]#
SDLE[3:0]#
INTCHIP0
ESC/ PCEB Interface
SPKR
SLOWH#
Timers
IRQ[15:9,7:3,1]
IRQ8#
INT
NMI
Interrupt Controller
SALE#]
LASAOE#
SALAOE#
FERR#
IGNNE#
LBIOSCS#
KYBDCS#
ALTRST#
ALTA20
ABFULL
RTCALE
RTCRD#
RTCWR#
FDCCS#
DSKCHG
DLIGHT#
CRAMRD#
CRAMWR#
XBUSTR#
XBUSOE#
GPCS[2:0]#/ECS[2:0]
AEN[4:1]/EAEN[4:1]
Integrated Support Logic
EISA Bus Interface
BCLKOUT
BCLK
LA[31:27]#/CPG[4:0]
LA[26:24]#
LA[23:2]
BE[3:0]#
M/IO#
W/R#
EX32#
EX16#
START#
CMD#
EXRDY
SLBURST#
MSBURST#
MASTER16#
SD[7:0]
BALE
SA[1:0]
SBHE#
M16#
IO16#
MRDC#
MWTC#
SMRDC#
SMWTC#
IORC#
IOWC#
CHRDY
IOCHK#
NOWS#
OSC
REFRESH#
RSTDRV
AEN#
DMA Controller w S-G
DREQ[7:5,3:0]
DACK[7:5,3:0]
EOP
EISA Arbiter
MREQ[7:4]#/PIRQ[0:3]#
MREQ[3:0]#
MACK[3:0]#/EMACK[3:0]
Test
TEST
290467–5

82375EB
PCI/EISA BRIDGE (PCEB)

- **Provides the Bridge between the PCI Bus and EISA Bus**
- **100% PCI and EISA Compatible**
  - **— PCI and EISA Master/Slave Interface**
  - **— Directly Drives 10 PCI Loads and 8 EISA Slots**
  - **— Supports PCI at 25 MHz to 33 MHz**
- **Data Buffers Improve Performance**
  - **— Four 32-Bit PCI-to-EISA Posted Write Buffers**
  - **— Four 16-Byte EISA-to-PCI Read/Write Line Buffers**
  - **— EISA-to-PCI Read Prefetch**
  - **— EISA-to-PCI and PCI-to-EISA Write Posting**
- **Data Buffer Management Ensures Data Coherency**
  - **— Flush Posted Write Buffers**
  - **— Flush or Invalidate Line Buffers**
  - **— Instruct All PCI Devices to Flush Buffers Pointing to PCI Bus before Granting EISA Access to PCI**
- **Burst Transfers on both the PCI and EISA Buses**
- **32-Bit Data Paths**
- **Integrated EISA Data Swap Buffers**
- **Arbitration for PCI Devices**
  - **— Supports Six PCI Masters**
  - **— Fixed, Rotating, or a Combination of the Two**
- **PCI and EISA Address Decoding and Mapping**
  - **— Positive Decode of Main Memory Areas (MEMCS# Generation)**
  - **— Four Programmable PCI Memory Space Regions**
  - **— Four Programmable PCI I/O Space Regions**
- **Programmable Main Memory Address Decoding**
  - **— Main Memory Sizes up to 512 MBytes**
  - **— Access Attributes for 15 Memory Segments in First 1 MByte of Main Memory**
  - **— Programmable Main Memory Hole**
- **Integrated 16-Bit BIOS Timer**
- **208-Pin QFP Package**
- **5V CMOS Technology**

The 82375EB PCI-EISA Bridge (PCEB) provides the master/slave functions on both the Peripheral Component Interconnect (PCI) Local Bus and the EISA Bus. Functioning as a bridge between the PCI and EISA buses, the PCEB provides the address and data paths, bus controls, and bus protocol translation for PCI-to-EISA and EISA-to-PCI transfers. Extensive data buffering in both directions increases system performance by maximizing PCI and EISA Bus efficiency and allowing concurrency on the two buses. The PCEB's buffer management mechanism ensures data coherency. The PCEB integrates central bus control functions including a programmable bus arbiter for the PCI Bus and EISA data swap logic for the EISA Bus. Integrated system functions include PCI parity generation, system error reporting, and programmable PCI and EISA memory and I/O address space mapping and decoding. The PCEB also contains a BIOS Timer that can be used to implement timing loops. The PCEB is intended to be used with the EISA System Component (ESC) to provide an EISA I/O subsystem interface.


PCEB Block Diagram
PCICLK
PCIRST#
AD[31:0]
C/BE[3:0]#
FRAME#
TRDY#
IRDY#
STOP#
PLOCK#
IDSEL
DEVSEL#
PAR
PERR#
PCI Interface
Data Buffers
EISA Address Decoder
EISA Interface
BCLK
START#
CMD#
M/IO#
W/R#
EXRDY
EX32#
EX16#
MSBURST#
SLBURST#
LOCK#
BE[3:0]#
LA[23:2]
LA[31:24]
REFRESH#
MEMCS#
PIODEC#
PCI Address Decoder
BIOS Timer
EISA Data Swap Logic
SD[31:0]
CPUREQ#
CPUGNT#
REQ1#
GNT1#/RESUME#
REQ0#/PCEBGNT#
GNT0#/PCEBREQ#
REQ[3:2]#
GNT[3:2]#
PCI Arbiter
PCEB/ ESC Interface
EISAHOLD
EISAHOLDA
PEREQ#/INTA#
NMFLUSH#
SDCPYEN01#
SDCPYEN02#
SDCPYEN03#
SDCPYEN13#
SDCPYUP
SDOE[2:0]#
SDLE[3:0]#
INTCHIP0
MEMREG#
FLSHREQ#
MEMACK#
TEST#
Test
ISA Interface
IO16#
290467-6

APPLICATION NOTE

A Memory Subsystem for the Intel486™ Family of Microprocessors including Second Level Cache

2

GREGORY A. ROBERTSON
SENIOR APPLICATION ENGINEER

March 1992

Order Number: 240799-002

A Memory Subsystem For The Intel486™ Family Of Microprocessors Including Second Level Cache

CONTENTS

PAGE

1.0 INTRODUCTION 2-865

2.0 THE 485TURBOCACHE SECOND LEVEL CACHE MODULE 2-865

3.0 PROCESSOR FEATURE REVIEW 2-869

3.1 The Burst Cycle 2-869

3.2 The KEN# Input 2-870

3.3 Bus Characteristics 2-871

4.0 DRAM INTERFACE OVERVIEW ... 2-874

4.1 Functional Blocks 2-874

4.2 Address Path Logic 2-875

4.3 Data Path 2-877

4.4 Second Level Cache Support 2-878

4.5 Control Logic 2-879

5.0 MEMORY SUBSYSTEM FUNCTION 2-881

5.1 CPU Interface Signals 2-881

5.2 Data Path Control 2-882

CONTENTS

PAGE

5.3 Address Path Control 2-882

5.4 DRAM Interface 2-882

5.5 Controller Signals 2-883

5.6 Read Cycles 2-883

5.7 Write Cycles 2-885

5.8 Consecutive Bus Cycles 2-887

5.9 Page Miss Cycles 2-888

5.10 Refresh Cycles 2-890

6.0 CONTROLLER IMPLEMENTATION 2-891

6.1 Cycle Tracking Logic 2-891

6.2 RAS# Logic 2-894

6.3 CAS# Logic 2-895

6.4 Write Control Logic 2-897

6.5 Burst Address Logic 2-897

7.0 SUMMARY 2-899

7.1 Timing Restrictions 2-899

APPENDIX A 2-900

This Application Note covers both the Intel486™ DX and Intel486™ SX Microprocessors. Refer to the respective datasheets for detailed specifications.

1.0 INTRODUCTION

The Intel486™ CPU contains several improvements over its predecessor, the highly successful Intel386™ CPU. One of the most important of these is the processor's data access rate. The Intel486 CPU can access instructions and data from its on-chip cache in the same clock cycle. To support the processor's redesigned internal data path, the external bus has also been optimized and can access external memory at twice the rate of the Intel386 CPU. The internal cache requires rapid access to entire cache lines. Invalidation cycles must be supported to maintain consistency with external memory. All of these functions must be supported by the external memory system. Without them, the full performance potential of the CPU cannot be attained.

The requirements of todays multitasking and multiprocessor operating systems also put increased demand on the external memory system. OS support functions such as paging and context switching can degrade reference locality. Without efficient access to external memory, the performance of these functions is reduced.

Second level caching is a technique used to improve the memory interface. Some applications, such as multiuser office computers, require this feature to meet performance goals. Single-user systems, on the other hand, may not warrant the extra cost. Given the variety of applications incorporating the Intel486 CPU, memory system architecture will be very diverse.

In this application note, we will work with an example to discuss the details of memory system design. In the example, we have supported as many functions of the CPU as possible. An optional second-level cache is included. A write buffer is also implemented to reduce write latency. The cache supports zero wait state read cycles. The DRAM controller supports the following devices with the wait states shown in Table 2. The DRAM speed given in Table 1 is the RAS access time (tRAC). Table 2 summarizes the bus clocks required for each function.

Table 1

CPU	Clock Frequency	DRAM Speed
Intel486 DX/ Intel486 SX	25 MHz	100 ns
Intel486 DX	33 MHz	70 ns

Many of the functions and optimizations included here will not be required in every application. The example provides guidelines for the hardware designer but will not necessarily provide the optimal cost/performance solution for many applications. For example, 11 PLDs are required to implement the memory control logic partially due to the implementation of a back-off capability. An address register must also be used to implement this function. If this function is not used, the control logic can be substantially reduced. These and other optimizations will be discussed in the summary.

Table 2

DRAM Function	First Access Burst	Subsequent Burst Accesses	Write Cycles
Page Hit	3	1	2
Page Miss	7	1	5*

NOTE:
*Write miss latencies occur only during cycles subsequent to a write miss cycle.

The discussion assumes a working knowledge of computer system design. Items discussed but not explained include DRAM operation, PLD programming and operation, worst-case timing analysis and Intel486 CPU bus operation. The complete schematics and PLD equations are in Appendix A.

2.0 THE 485TURBOCACHE SECOND LEVEL CACHE MODULE

Several different types of second level cache architectures are possible candidates for use with the Intel486 CPU. For single CPU systems, the different architectures offer similar performance benefits in most cases. The reason they are so similar is the mechanism which improves performance. The primary benefit of the second level cache is bus cycle latency reduction.

In most systems which incorporate a single Intel486 CPU, bus traffic from other bus masters is minimal. With any reasonable memory system the CPU uses at most 50% to 70% of the bus. Therefore reduction of bus cycle latency is the only performance benefit external logic can offer.

The second level cache used in this example is an economical method of reducing read cycle latency. The 485Turbocache module contains the control circuits, data and tag ram required to implement a 64k or 128k byte second level cache for Intel486 CPU (25 MHz/ 33 MHz). It is organized as a two way set associative cache. In this example, the 128k byte size cache module is used.

One of the most interesting aspects of this device is it can be a system option. To provide this capability the device is configured as a look-aside cache. It monitors the CPU address and control signals. When a cycle occurs in which the cache can supply data, it intervenes. The cache module then supplies an entire 16-byte line with no wait states.

The performance improvement offered by this cache is substantial in some environments. This performance improvement is particularly obvious when executing multitasking, multiuser operating systems such as UNIX and OS/2. Some users, however, may not require the performance improvement offered by the cache. In these cases, the cache as an option is attractive.

By designing the cache subsystem as an option both user's requirements can be met. A single system design can be manufactured for both customers. The UNIX or OS/2 user can add the cache module. Other users may or may not require the module. They can choose the system configuration which meets their price-performance needs.

When a 485Turbocache Module is connected to an Intel486 processor system, the processor's internal cache should map the entire address space including that of the 485Turbocache Module devices to provide the highest performance. This is the most efficient configuration. The Intel486 CPU can access a line from its internal cache in one clock and the 485Turbocache Module provides the next fastest access in two clocks for the first doubleword and the remaining three doublewords in three clocks.

The processor's address bits A2–A31 are connected to A2–A31 on the 485Turbocache Module. Internally, address bits A4–A15 are sent to both sets, to select one of 4,096 locations. Because the cache is two-way set associative, each address points to information stored in two banks. On each read or write cycle, the value of A16–A31 is compared to the tags stored at the location addressed by A4–A15. If they are equal, and if the valid bit is set, then a hit occurs. If a read cycle is in progress, then the 485Turbocache Module returns data to the Intel486 CPU. If the hit cycle is a write cycle, then the new data is updated in the 485Turbocache Module.

The BRDY0# output and the CBRDY# input must be used in forming of the Intel486 CPU's BRDY# input. Similarly, the CRDY# input must be used in forming of the Intel486 CPU's RDY# input.

The memory system generates KEN# to the Intel486 CPU when read data needs to be cached. The 485Turbocache Module receives this signal as the SKEN# input. The 485Turbocache Module's CKEN# output can be used in the formation of the KEN# input to the Intel486 CPU. CKEN# can be used in conjunction with other logic that can deassert KEN# to the CPU when the system wants the current line fill to be cached by the 485Turbocache Module and not cached in the Intel486 CPU.

The 485Turbocache Module connects directly to the Intel486 CPU's address lines A2–A31. The designer may have to add external buffers to the address outputs, depending upon the loading. Other signals connected to the Intel486 CPU include the burst control signals, the bus cycle definition signals, the byte enables, the ADS# signal, and the data and parity signals. The 485Turbocache Module and CPU connections are shown in Figure 1. The 485Turbocache Module main memory controller and bus controller interface are shown in Figure 2.

Read Hit Cycles

A read hit cycle occurs when requested data is present in the 485Turbocache Module. The Intel486 CPU attempts to retrieve the entire line from the 485Turbocache Module without incurring wait states. This may be accomplished by activating the KEN# input at the end of T1 (the clock in which ADS# becomes active). There is very little time to decode the address, generate the KEN# signal to the Intel486 CPU, and complete a zero wait state read operation. Because KEN# is sampled twice, it is possible to always assert KEN# in T1 and to wait until the end of a line fill to decide whether the data is cacheable. (See Section 3.2.)

CKEN# is used in the formation of the KEN# signal to the Intel486 CPU. CKEN# is activated in T1 (see Figure 3 and Figure 4). If a read hit occurs, data can be sent to the Intel486 CPU in zero wait states and can still be cached in the processor's on-chip cache. The 485Turbocache Module asserts CKEN# which remains asserted for the duration of the read hit cycle (unless WPSTRP# is low and the line is write protected). This means that the Intel486 CPU will cache the entire line unless external logic is added to cause the KEN# signal to be sampled high in the clock before the last BRDY0# from the 485Turbocache Module.

If the CKEN# input from the 485Turbocache Module is connected directly to the KEN# input of the Intel486 CPU, then the CPU will sample KEN# active at the end of T1. To deassert KEN# to the processor, the system must create another signal that is used in the formation of the Intel486 CPU's KEN#, and the 485Turbocache Module's SKEN#. Using this technique, a non-cacheable memory read cycle can be performed. In addition, if the 485Turbocache module's write protection feature is to be used, KEN# and SKEN# must be generated separately since the CPU does not support write protection.

The BRDY# signal to the Intel486 CPU can be generated from many sources. Therefore, the various signals

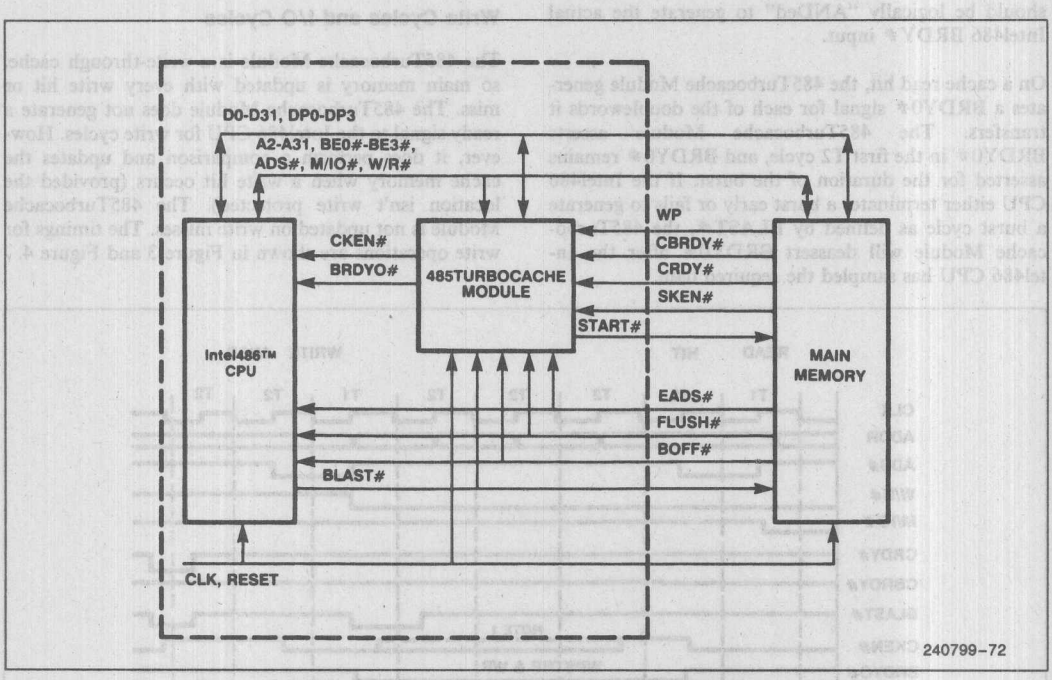


Figure 1. 485Turbocache Module and Intel486™ CPU Connections

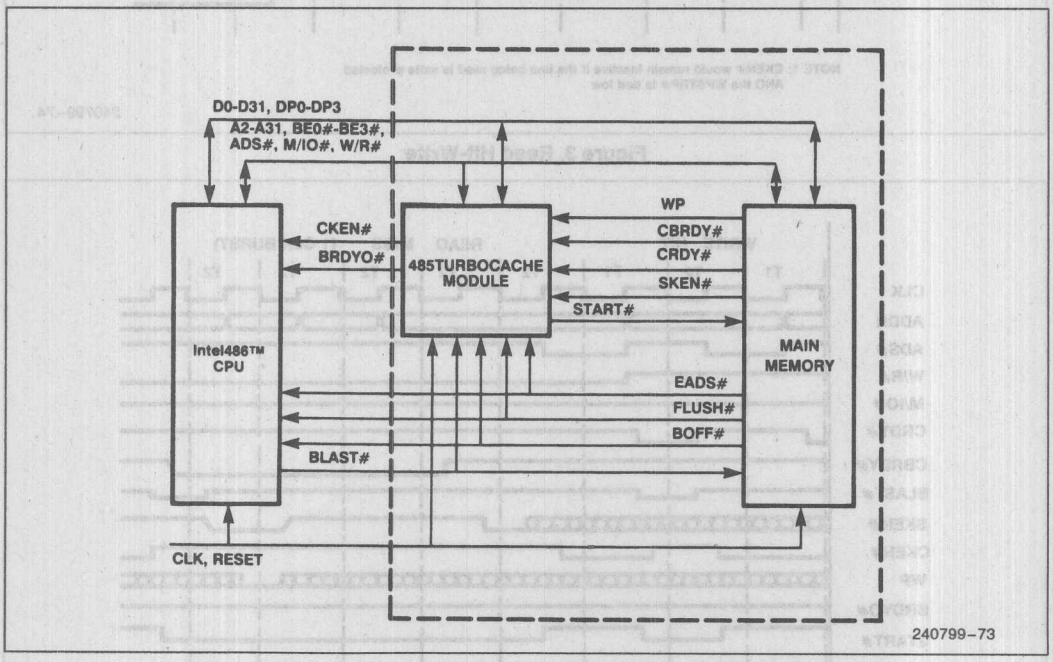


Figure 2. 485Turbocache Module and Main Memory Connections

should be logically "ANDed" to generate the actual Intel486 BRDY# input.

On a cache read hit, the 485Turbocache Module generates a BRDY0# signal for each of the doublewords it transfers. The 485Turbocache Module asserts BRDY0# in the first T2 cycle, and BRDY0# remains asserted for the duration of the burst. If the Intel486 CPU either terminates a burst early or fails to generate a burst cycle as defined by BLAST#, the 485Turbocache Module will deassert BRDY0# after the Intel486 CPU has sampled the required data.

Write Cycles and I/O Cycles

The 485Turbocache Module is a write-through cache, so main memory is updated with every write hit or miss. The 485Turbocache Module does not generate a ready signal to the Intel486 CPU for write cycles. However, it does perform a comparison and updates the cache memory when a write hit occurs (provided the location isn't write protected). The 485Turbocache Module is not updated on write misses. The timings for write operations are shown in Figure 3 and Figure 4.

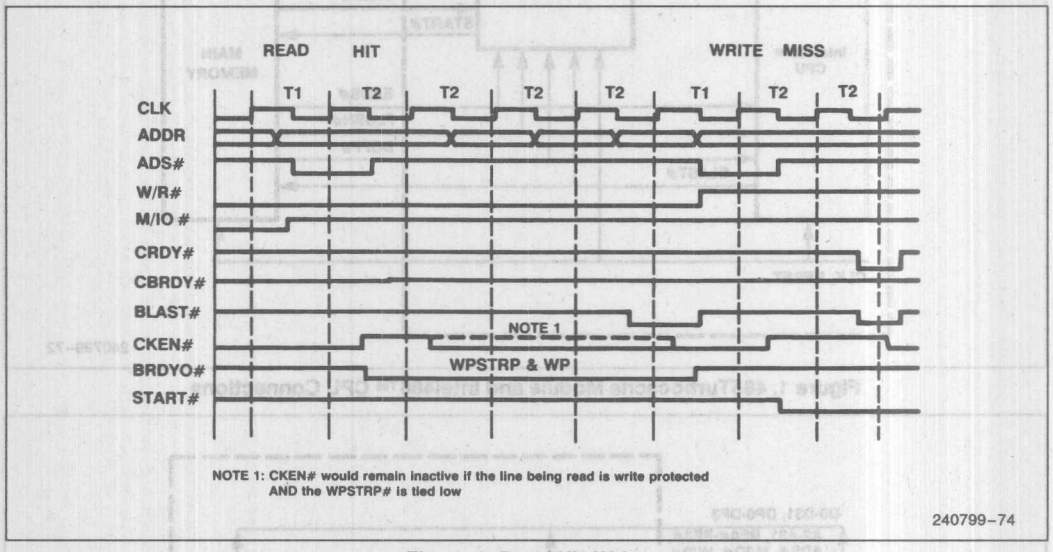


Figure 3. Read Hit-Write

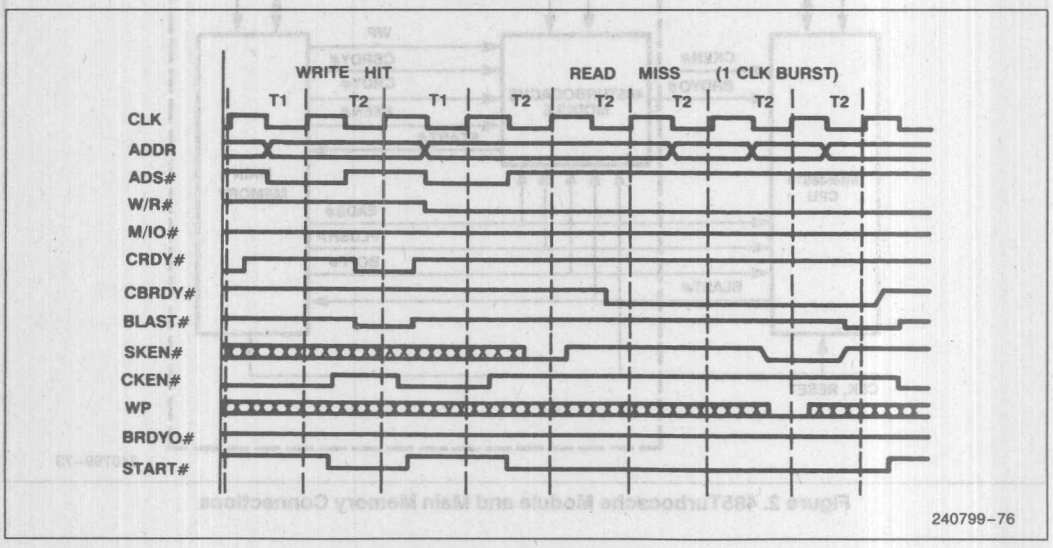


Figure 4. Write-Read Miss

Because the 485Turbocache Module is a write-through cache, writes are immediately forwarded to the system. If a processor write occurs on a valid entry that is not write protected, the new data will be stored into the cache in zero wait states. The 485Turbocache Module will not generate a ready signal. It is the systems's responsibility to update the system memory on all writes and to terminate all cycles with a ready signal. Even after the 485Turbocache Module has completed its internal write update, it remains idle until the system returns a ready to the processor.

A cache location can be write protected by asserting the WP input to the 485Turbocache Module. The WP signal must be valid during the third BRDY0# or RDY# of a cache line fill cycle until the end of the cycle. It sets a state bit for a particular cache location and remains in effect until the line is invalidated. Tieing WPSTRP# low will prohibit the write protected entry from being cached by the Intel486 CPU in subsequent accesses. The entry can be invalidated by any of the following: a flush operation, a reset operation, an invalidation cycle, or an LRU replacement.

When an Intel486 CPU cycle produces a write hit to a write-protected 485Turbocache Module location, data in the cache is not modified. The 485Turbocache Module responds in the same way whether or not a write hit location is write protected by asserting the START# signal. It is the designer's responsibility to prevent inconsistencies between the 485Turbocache Module and main memory when using the WP signal.

The 485Turbocache Module ignores all I/O cycles. When an I/O cycle is executed by the Intel486 processor, the system responds and terminates the cycle. The 485Turbocache Module does not assert the START# signal for I/O accesses, and the system should monitor the M/IO# signal rather than wait for the assertion of the START# signal.

System Cacheability Indication

The 485Turbocache Module uses the cache enable scheme of the Intel486 CPU. A cache update to the 485Turbocache Module requires activating the SKEN# signal. The signal is sampled twice, first to determine if a line fill cycle should be run and second to confirm that the line fill should actually be cached. SKEN# is ignored during write cycles.

Typically, the system will use the same logic to generate the Intel486 CPU's KEN# signal and the 485Turbocache Module requires activating the SKEN# signal. However, it is not necessary for both to be asserted during an access. It is possible to use different cacheing maps for the CPU cache and the 485Turbocache Module cache because the Intel486 CPU and the 485Turbocache Module maintain their own cache contents via snooping.

3.0 PROCESSOR FEATURE REVIEW

The improvements made to the CPU bus interface obviously impact the memory subsystem design. It is important to understand the impact of these features before attempting to define the system. This section is a review of the bus features which affect the memory interface. The features and their impact on memory system design is discussed.

3.1 The Burst Cycle

The Intel486 CPU's burst bus cycle feature has more impact on the memory logic than any other feature. It is the most significant departure from previous bus architectures. A large portion of the control logic is dedicated to supporting this feature. The second level cache is also primarily dedicated to supporting burst cycles.

To understand why the logic is designed this way, we must first understand the function of the burst cycle. Burst cycles are generated by the CPU if, and only if, two events occur. First, the CPU must request a cycle which is longer in bytes than the data bus can accomodate. Second, the BRDY# signal must be activated to terminate the cycle. When these two events occur a burst cycle will take place. Note that this cycle will occur regardless of the state of the KEN# input. The KEN# input's function is discussed in the next section.

With this definition we see that several cases are included as "burstable". Some examples of burstable cycles are listed in Table 3. These cycle's length is shown in bytes to clarify the case listed.

Table 3

Burst Bus Cycle	Size (bytes)
All Code Fetches	16
Descriptor Loads	8
Cacheable Reads	16
Floating Point Operand Loads	8
Bus Size 8(16) Writes	4 (max)

The last case shows that write cycles are burstable. In this case a write cycle is transfered on an 8 or 16 bit bus. If BRDY# is returned to terminate this cycle the CPU will generate another without activating ADS#.

Using the burst write feature has debatable performance benefit. Some systems may implement special functions which benefit from the use of burst writes. However, the Intel486 CPU does not write cache lines. Therefore, all write cycles are 4 bytes long. Also, most of the devices which use dynamic bus sizing are read only. This fact further reduces the utility of burst writes.

Due to these facts, the design example used here does not implement burst write cycles. In fact, the BRDY# input is only asserted during main memory read cycles and cache hit cycles. RDY# is used to terminate all memory write cycles. RDY# is also used for all cycles which are not in the memory subsystem or are not capable of supporting burst cycles. The RDY# input is used, for example, to terminate an EPROM or I/O cycle.

3.2 The KEN# input

The primary purpose of the KEN# input is to determine whether a cycle is to be cached. Only read data and code cycles can be cached. Therefore, these cycles are the only cycles affected by the KEN# input.

Figure 5 shows a typical burst cycle. In this sequence the value of KEN# is important in two different places. First, to begin a cacheable cycle KEN# must be active the clock before BRDY# is returned. Second, KEN# is sampled the clock before BLAST# is active. At this time the CPU determines whether this line will be written to the cache.

The state of KEN# also determines when read cycles can be bursted. Most read cycles are initiated as 4 byte long from the CPU's cache unit. When KEN# is sampled active the clock before BRDY# or RDY# is returned, the cycle is converted to a 16 byte cache line fill by the bus unit. This way, a cycle which would not have been bursted can now be bursted by activating BRDY#.

Some read cycles can be bursted without activating KEN#. The most prevalent example of this type of read cycle is code fetches. All code fetches are generated as 16-byte cycles from the CPU's cache unit. So, regardless of the state of KEN#, code fetches are always burstable. In addition, several types of data read cycles are generated as 8-byte cycles. These cycles, mentioned previously, are descriptor loads and floating point operand loads. These cycles can also be bursted at any time.

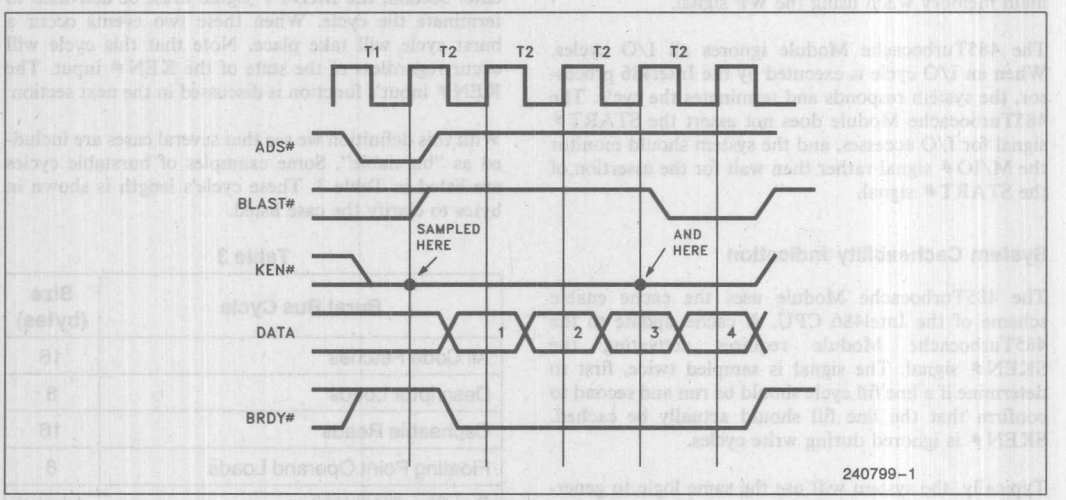


Figure 5. Typical Burst Cycle

It's obvious that the use of the KEN# input affects performance. The design example used here illustrates one way to use this signal effectively.

The primary concern when using KEN# is generating it in time for zero wait state read cycles. Most main memory cycles will be zero wait state if a second level cache is implemented. In this example, the main memory is one wait state during most read cycles. Any Cache access will take place with zero wait states. KEN# must, therefore, be valid during the first T2 of any read cycle.

Once this requirement is established, a problem arises. Decode functions are inherently asynchronous. Therefore, the decoded output which generates KEN# must be synchronized. If not, the setup and hold times of the CPU will be violated and internal metastability will result. With synchronization, the delay required to generate KEN# will be at least three clocks. In this example 4 clocks are required. In either case the KEN# signal will not be valid before BRDY# is returned for zero or one wait state cycles.

This problem is resolved if KEN# is made normally active. Figure 6 illustrates this function. In this diagram KEN# is active during the first two clocks of the burst cycle. If this is a data read cycle, KEN# being active at this time causes it to be converted to a 16 byte length. The decode and synchronization of KEN# takes place during the first two T2 states of the cycle. If the cycle turns out to be non-cacheable, KEN# will be deactivated in the third T2. Otherwise KEN# will be left active and the data retrieved will be written to the cache.

Some memory devices may be slow enough that 16-byte cycles are undesireable. In this case more than three wait states will exist. The KEN# signal can be deactivated prior to returning RDY# or BRDY# if three or more wait states are present. As a result these slow cycles will not be converted to 16-byte cache line fills.

3.3 Bus Characteristics

The internal cache causes other effects which impact the memory subsystem design. Perhaps the most obvious of these is the effect on bus traffic. The fact that the internal cache uses the write-through policy dramatically increases the number of write bus cycles. Figure 7 illustrates this effect. The top chart shows the bus cycle mix for an application executed with the Intel386DX CPU. The bottom chart shows the same application executed with the Intel486 CPU. The percentage of write bus cycles jumps to 70% from 30% when this application is executed with the Intel486 CPU.

It seems intuitively obvious that many of these write cycles would be consecutive. In fact, 70% of all write cycles are consecutive. Furthermore, 50% of all write cycles occur three in a row. It is obvious from these statistics that optimizing the memory subsystem for write cycles can improve performance. But it is important to optimize the memory system for consecutive write cycles. Improving individual write cycle latency will not buy much performance if subsequent write cycles suffer.

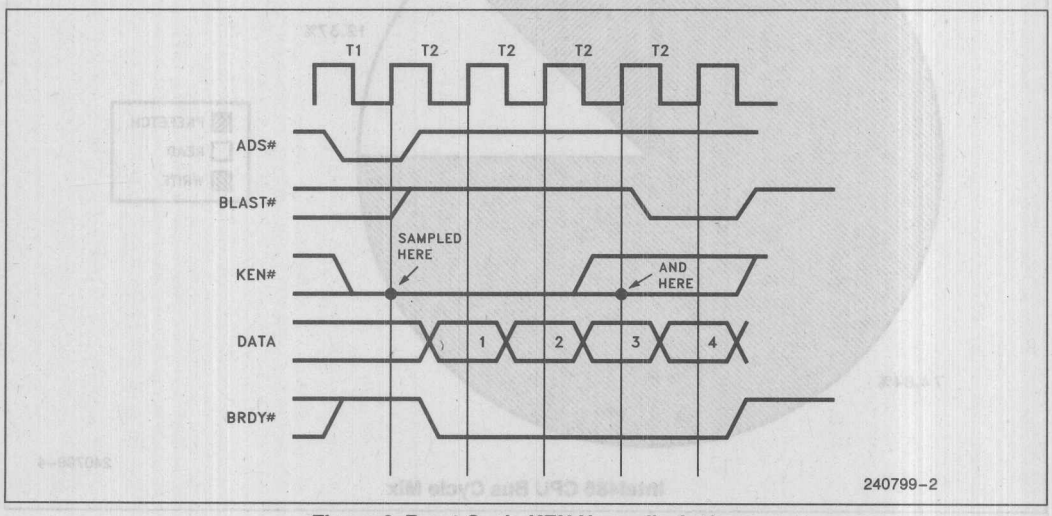
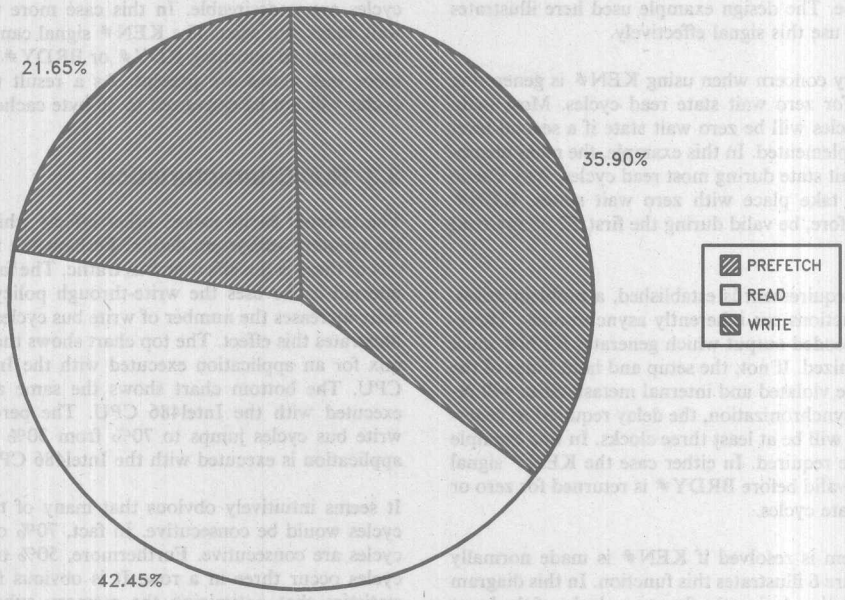
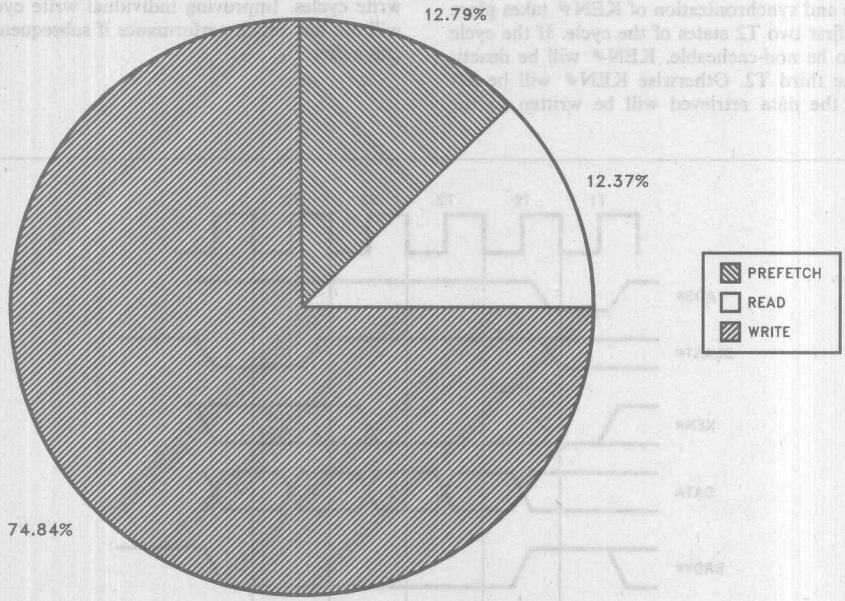


Figure 6. Burst Cycle KEN Normally Active



Intel386DX CPU Bus Cycle Mix



Intel486 CPU Bus Cycle Mix

Figure 7. CPU Bus Cycle Mix

A technique called write posting proves ideal for this purpose. This technique allows consecutive write cycles to be overlapped. It also allows write cycles to be overlapped with second level cache cycles and reduces overall write miss latency.

Using the write posting technique adds complexity to the system logic. It is therefore valid to ask what performance improvement is gained by using this technique. This question is especially pertinent when we consider the logic already implemented in the Intel486 CPU to improve write performance. The internal Intel486 write buffers decouple the processor execution unit from the external bus.

Analysis has shown that, in general, 6% degradation in performance can be expected for every additional wait state added to write cycles. This analysis was performed by measuring the CPU clocks required to execute several applications.

The same analysis has shown that write posting reduces average write latency to 2.5 clocks. Without write posting average write latency is 4 clocks. From this data we can conclude that approximately 9% performance improvement can be obtained by using write posting. This improvement may increase due to other affects. These affects, such as overlapping write cycles with cache reads, are discussed in subsequent sections.

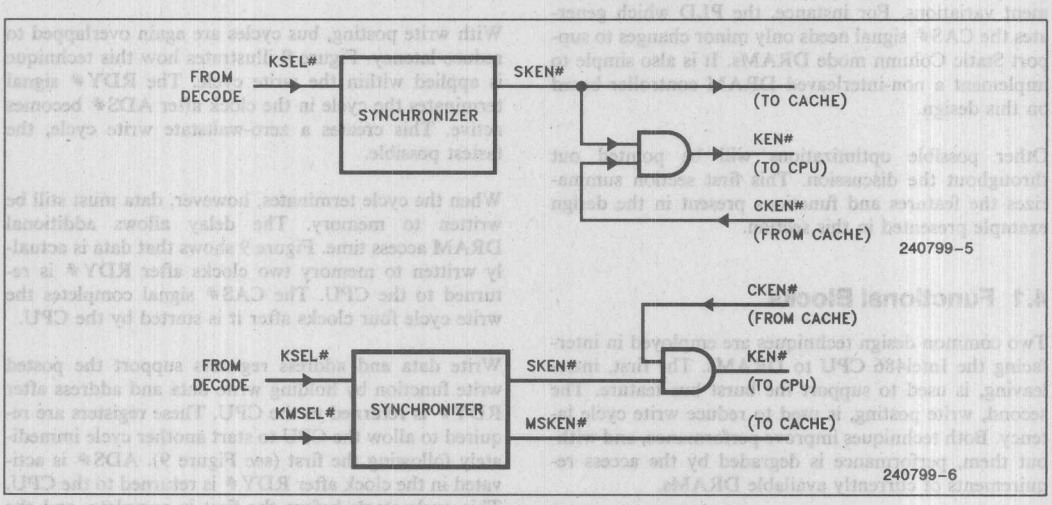


Figure 8. KEN# Logic for Second-Level Cache

4.0 DRAM INTERFACE OVERVIEW

The Intel486 CPU bus interface unit integrates several functions which improve the memory access rate. These features must be supported by the memory subsystem to provide the intended performance benefit. They are supported by the memory subsystem example. The example also includes logic support for a second-level cache. An overview of the subsystem is presented in this section. Details of the function and logic design of this subsystem are presented in later sections.

This subsystem follows a modular design. Only minor changes to particular logic sections are needed to implement variations. For instance, the PLD which generates the CAS# signal needs only minor changes to support Static Column mode DRAMs. It is also simple to implement a non-interleaved DRAM controller based on this design.

Other possible optimizations will be pointed out throughout the discussion. This first section summarizes the features and functions present in the design example presented in this section.

4.1 Functional Blocks

Two common design techniques are employed in interfacing the Intel486 CPU to DRAMs. The first, interleaving, is used to support the burst bus feature. The second, write posting, is used to reduce write cycle latency. Both techniques improve performance, and without them, performance is degraded by the access requirements of currently available DRAMs.

Interleaving can be implemented in several ways. Here, alternate 32-bit DRAM banks are accessed.The bank accessed is determined by the value of A2. In this way, even DWORDs (A2 = 0) are stored in one bank while odd DWORDs (A2 = 1) are stored in the other. When data is retrieved from memory during a cache line fill, cycles are overlapped to allow single clock DWORD accesses. Timing of this operation is detailed in the next section.

A multiplexor alternates data flow between the DRAM banks and the appropriate data path is selected according to the value of A2. The multiplexor prevents bus contention.

With write posting, bus cycles are again overlapped to reduce latency. Figure 9 illustrates how this technique is applied within the write cycle. The RDY# signal terminates the cycle in the clock after ADS# becomes active. This creates a zero-waitstate write cycle, the fastest possible.

When the cycle terminates, however, data must still be written to memory. The delay allows additional DRAM access time. Figure 9 shows that data is actually written to memory two clocks after RDY# is returned to the CPU. The CAS# signal completes the write cycle four clocks after it is started by the CPU.

Write data and address registers support the posted write function by holding write data and address after RDY# is returned to the CPU. These registers are required to allow the CPU to start another cycle immediately following the first (see Figure 9). ADS# is activated in the clock after RDY# is returned to the CPU. This cycle starts before the first is complete, and the cycles overlap by two clocks.

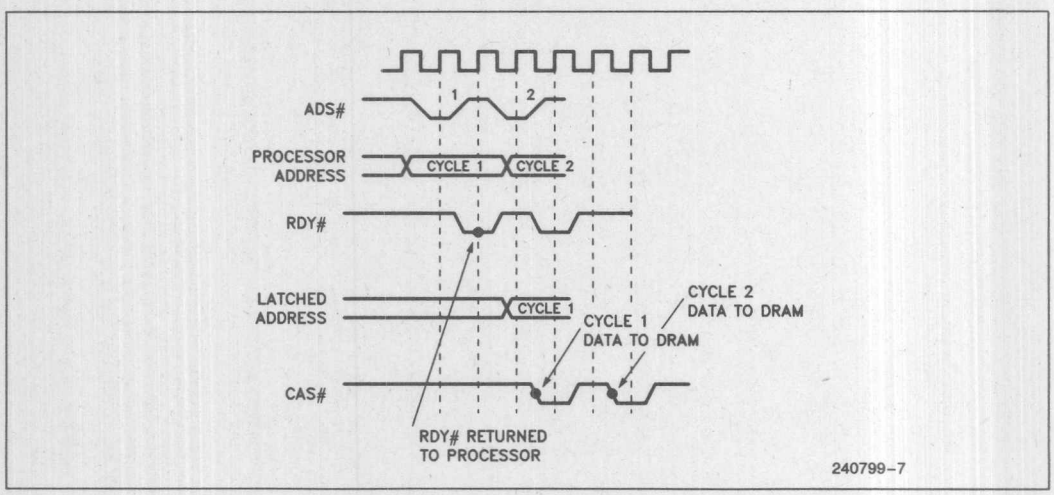


Figure 9. Write Posting

In effect the write cycle completes in two clocks. Write cycles can be overlapped in this manner indefinitely. The timing and logic required to support this function is described in Section 5.3.

Address registers also support invalidation with the AHOLD signal. They are required if AHOLD is activated when bus cycles are in progress to hold the current address while the bus cycle completes.

The efficient CPU interface and invalidation support make this DRAM subsystem well-suited for use with an optional cache. The memory system includes specific functions designed to support the optional 485Turbocache module. The subsystem supports 256K × 4 and 1Mbyte × 1 DRAM configurations. The minimum memory configuration is 2 Mbytes with 256K × 4 devices; the maximum is 16 Mbytes with 1Mbyte × 1 devices. Additional banks can be added to increase the memory capacity.

The control logic for this example is implemented with EPLDs.The modular approach allows quick modification so that the example can be tailored for specific implementation requirements.

The control state machine is distributed among the various EPLDs, and each functional block receives control input from other blocks. In addition most of the functional blocks are implemented as state machines.

Figure 10 is a top level block diagram of the memory system. This diagram depicts the sections of logic that will be described subsequently. We will first discuss the address path logic.

4.2 Address Path Logic

Unlike processors without on-chip caches, the address bus of the Intel486 processor is bidirectional. The address pins serve as inputs whenever external memory is changed by DMA or another CPU. The address is driven into the CPU to invalidate the corresponding cache entry if present.

Invalidation of the Intel486 CPU's internal cache can be performed in several different ways. This example supports invalidation cycles during a memory access.

As described in the previous section, AHOLD is used to perform the invalidation function. AHOLD tristates the Intel486 address bus. Address registers must be used to hold the address to allow the current bus cycle to be completed. These registers hold the current address when AHOLD is activated.

The registers shown in Figure 11 hold the entire row and column address, as well as the current byte enables and control definition. These signals are latched at the rising clock edge of the first T2 of a bus cycle. They must be held from this edge to allow zero wait state write cycles.

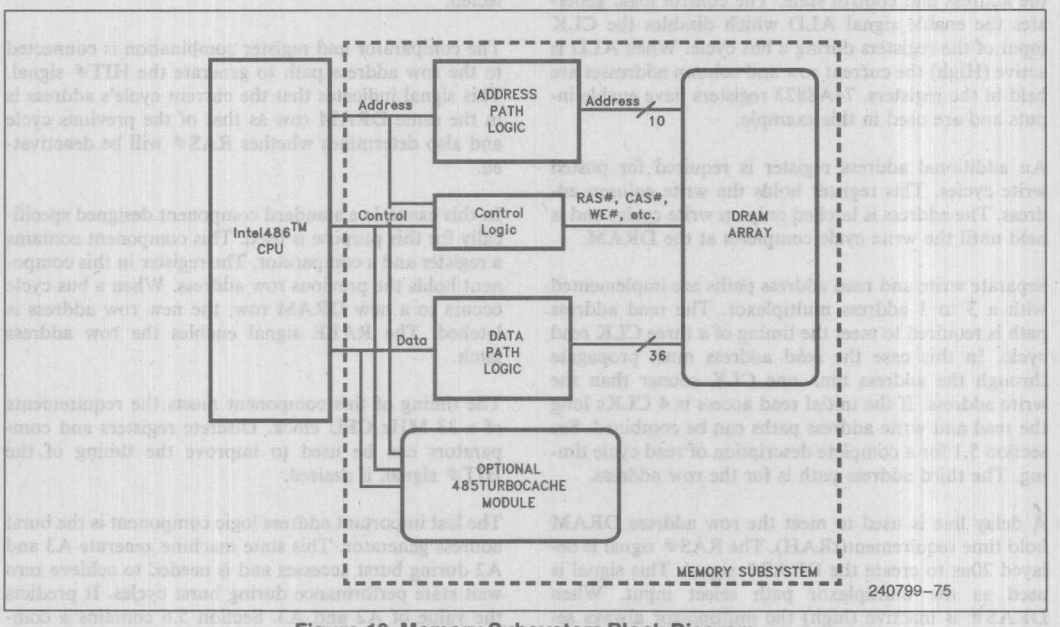


Figure 10. Memory Subsystem Block Diagram

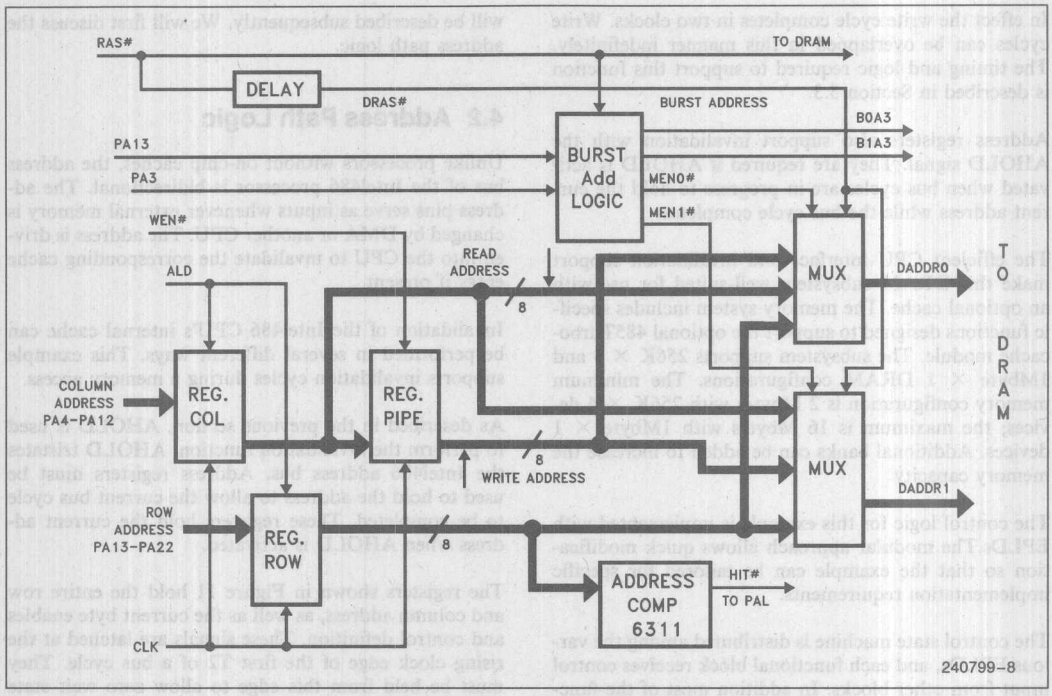


Figure 11. Address Path Logic

Registers with enable inputs are needed. The enable input can select the CLK edge appropriate for latching the address and control state. The control logic generates the enable signal ALD which disables the CLK input of the registers during a bus cycle. When ALD is active (High) the current row and column addresses are held in the registers. 74AS823 registers have enable inputs and are used in this example.

An additional address register is required for posted write cycles. This register holds the write column address. The address is latched only on write cycles and is held until the write cycle completes at the DRAM.

Separate write and read address paths are implemented with a 3 to 1 address multiplexor. The read address path is required to meet the timing of a three CLK read cycle. In this case the read address must propagate through the address mux one CLK sooner than the write address. If the initial read access is 4 CLKs long the read and write address paths can be combined. See section 5.1 for a complete description of read cycle timing. The third address path is for the row address.

A delay line is used to meet the row address DRAM hold time requirement(tRAH). The RAS# signal is delayed 20ns to create the DRAS# signal. This signal is used as the multiplexor path select input. When DRAS# is inactive (high) the multiplexor always selects the row address path. When DRAS# is active (low) the mux enable signal (MEN0# or MEN1#) controls whether the read path or the write path is selected.

The comparator and register combination is connected to the row address path to generate the HIT# signal. This signal indicates that the current cycle's address is in the same DRAM row as that of the previous cycle and also determines whether RAS# will be deactivated.

In this example a standard component designed specifically for this purpose is used. This component contains a register and a comparator. The register in this component holds the previous row address. When a bus cycle occurs to a new DRAM row, the new row address is latched. The RALE signal enables the row address latch.

The timing of this component meets the requirements of a 33 MHz CPU clock. Discrete registers and comparators can be used to improve the timing of the HIT# signal, if desired.

The last important address logic component is the burst address generator. This state machine generate A3 and A2 during burst accesses and is needed to achieve zero wait state performance during burst cycles. It predicts the value of A2 and A3. Section 5.6 contains a complete description of the burst cycle timing.

Note that because interleaving is used, A3 is the lowest order DRAM address. Two A3 equivalent signals are generated. One for Bank 0 (B0A0) and one for Bank 1 (B1A0). These signals are connected directly to the DRAM devices to meet critical timing requirements. The signals must also reflect the lowest order row address during miss cycles. As a result A13 is, therefore, an input to this logic. It is the lowest order row address when 1MBx1 DRAMs are used.

4.3 Data Path

A2 must also be predicted during burst read accesses. For this purpose, the burst address logic creates the DATASEL signal. DATASEL reflects the value of A2 for each access of a burst cycle and is used to control the data multiplexor as shown in Figure 12.

During burst cycles, the data multiplexor alternates between the bank 0 and bank 1 data paths. A2 must alternate states each clock for interleaving to function properly. The Intel486 CPU's burst address sequence is defined such that A2 changes state on every access.

A2 also selects the bank to which data is written. Data path logic is not involved in steering data during writes. Figure 12 shows separate data registers for each bank. Separate registers are only required to divide the data paths. These registers hold the same write data on every write cycle. The CAS# and WE# (write enable) signals control doubleword and byte steering.

Because of write data timing, the data registers must have the enable function. This function, can be used to select the clock upon which data is latched. The processor clock can be used as the register clock input to guarantee proper data setup and hold times.

As Figure 12 indicates, the MRDY# signal enables the write data registers and terminates memory write cycles. Data is therefore latched during the last clock of any write cycle.

MRDY# is restricted to write cycles while the MBRDY# signal is used for read cycles. The need for these signals illustrates the convenience of the CPU's dual-ready inputs. The MBRDY# signal enables the output of the data path multiplexor to prevent bus contention.

These ready signals are combined with similar system logic signals to form the processor RDY# and BRDY# inputs. I/O, peripheral and other non-burst devices can use the RDY# input. Burst devices, such as a second level cache controller must also use the BRDY# input. The MBRDY# and MRDY# signals are, therefore, used only with the DRAM control logic. They are isolated from the rest of the system by combinatorial logic.

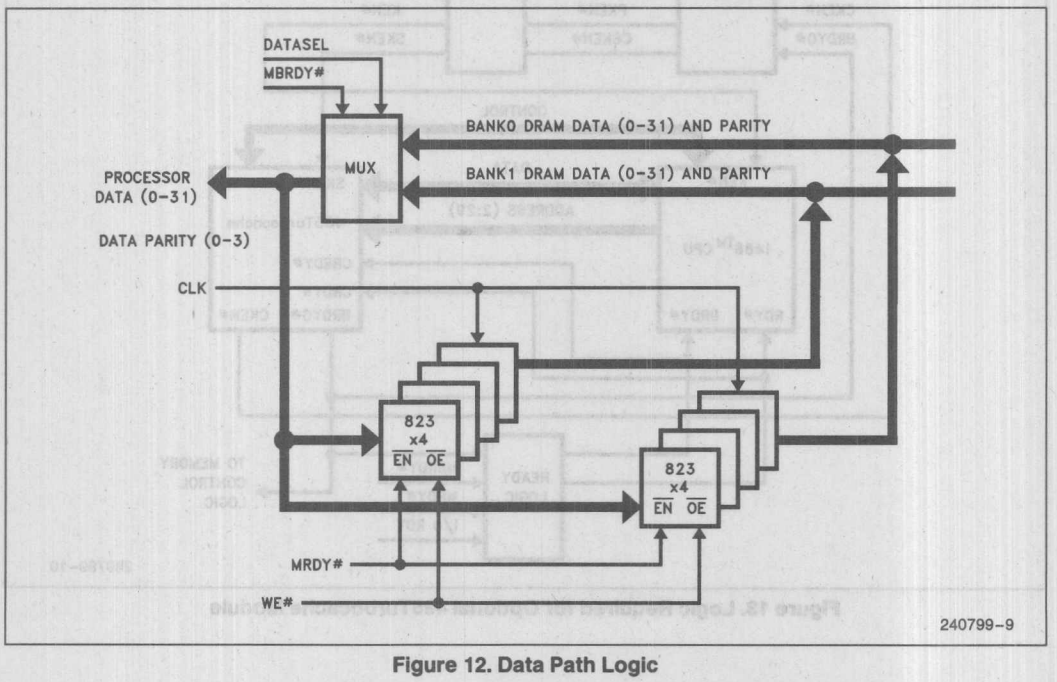


Figure 12. Data Path Logic

4.4 Second Level Cache Support

Second level cache strategies for the Intel486 CPU are diverse and application dependent. The example described illustrates a second level cache strategy that is ideal for single CPU systems.

The 485Turbocache second level cache used in this example is optional and is used to complement the Intel486 internal cache to improve the performance when running complex applications and operating systems. Some users will not require the extra performance. Since the cache is optional, O.E.M.'s or end-users can decide whether it should be included. System board design and manufacturing costs are thus eased since one system board supports multiple performance requirements.

The 485Turbocache is a completely self contained cache module. Optionality is accomplished by including control logic, tag ram and data ram in one package. A socket is added to the system board in much the same manner as a math coprocessor socket. In systems which, for example, run UNIX, the cache module is simply plugged in.

This option must, of course, be supported by the system logic. Specifically, the memory control logic is directly interfaced to the cache module. The DRAM controller example described here is particularly well- suited for this cache configurations.

The support included in the 485Turbocache module's memory control logic for the 485Turbocache module is illustrated in Figure 13. Since the 485Turbocache is a write-through cache, provision must be made for read cycles. When read data is found in the second level cache, the cycle is called a cache hit. At the time this cycle is determined to be a cache hit, it has already been started in the DRAM controller. This cycle must be aborted by the DRAM controller.

The BRDYO# signal from the 485Turbocache module provides a convenient cache hit indication. This signal is included in the decoder function. When a cache hit occurs, the DRAM controller aborts the cycle. The memory chip select signal is not activated and the first level control logic is reset aborting the cycle. The control logic then waits for another cycle to start. This function is very similar to the back-off function.

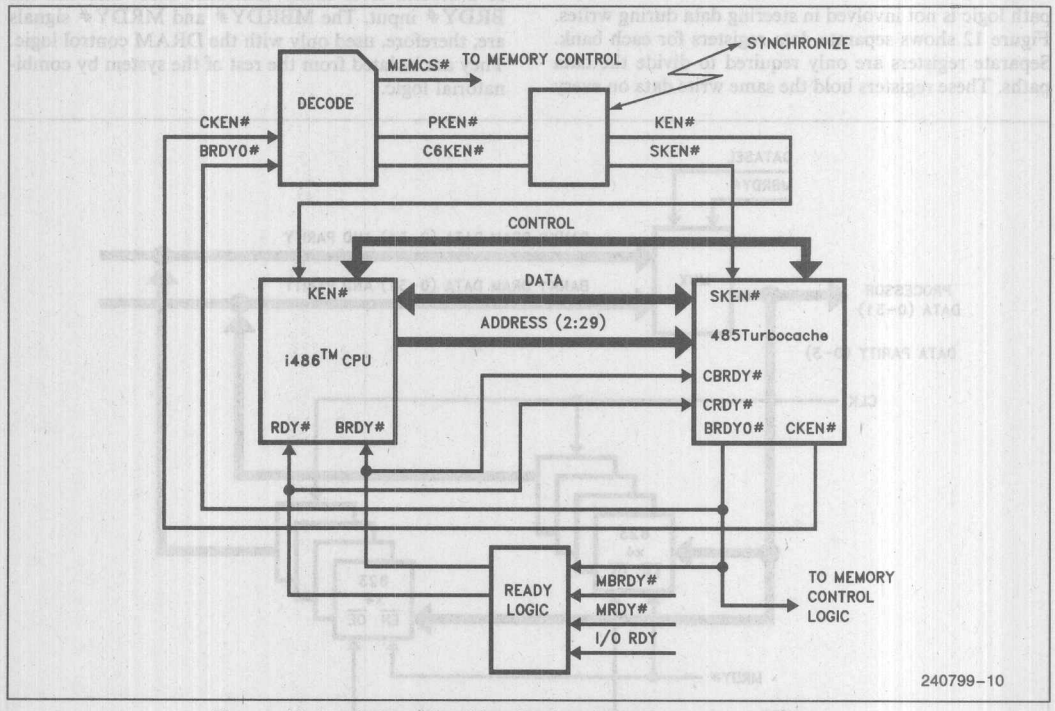


Figure 13. Logic Required for Optional 485Turbocache Module

Like the Intel486 internal cache, the 485Turbocache module supports non-cacheable memory by decoding. The SKEN# input is similar to the Intel486 CPU's KEN# input. This function is also supported by the decode logic. Note that, as with the KEN# signal, SKEN# must be synchronized to the CPU clock.

Separate cache enable inputs also allow areas of memory to be noncacheable in the Intel486 CPU internal cache yet cachable in the second level cache. This feature is convienient for BIOS.

4.5 Control Logic

Memory control logic generates the signals that control the memory devices, multiplexors, and registers described earlier. These control signals can be generated in a variety of ways. This example employs a distributed state machine.

Since this example is a prototype, PLDs were the logical choice for the controller implementation. Because the number of terms in a PLD is limited, the state machine implementation must be distributed. Function distribution was determined based on this constraint. Figure 14 shows a block diagram of the controller, with each block made up of one or two PLDs.

There are two levels of logic in the controller shown in Figure 14. The first is made up of two PLDs, one which tracks bus cycles and another which generates the MRDY# signal. The first level signals to PLDs in the second level that a cycle has started. The second level is made up of several PLDs which generate the actual control signals such as RAS# and CAS#.

Implementing the controller in this manner has two important advantages. First, more decode time is allowed. The cycle start signal, CIP#, is used by the second level logic to sample the decode output. CIP# is valid in the first T2 of any bus cycle. As a result, decode does not need to be valid until the end of this T2 bus state. Without this function, the decode output must be valid at the end of every T1 bus state. In this case, the time allowed for decode at 33 MHz is very short. With 7-ns PLDs, the time allowed for decode would be 7ns. With 5-ns PLDs, this time is still only 9ns. The advantage of the extra clock period is clear.

The second advantage of the two level approach is similarly clear. The CIP# signal indicates the start of a bus cycle to all second-level PLDs. Without this signal ADS# would have to be connected to these devices, and the resulting load on ADS# would be prohibitive.

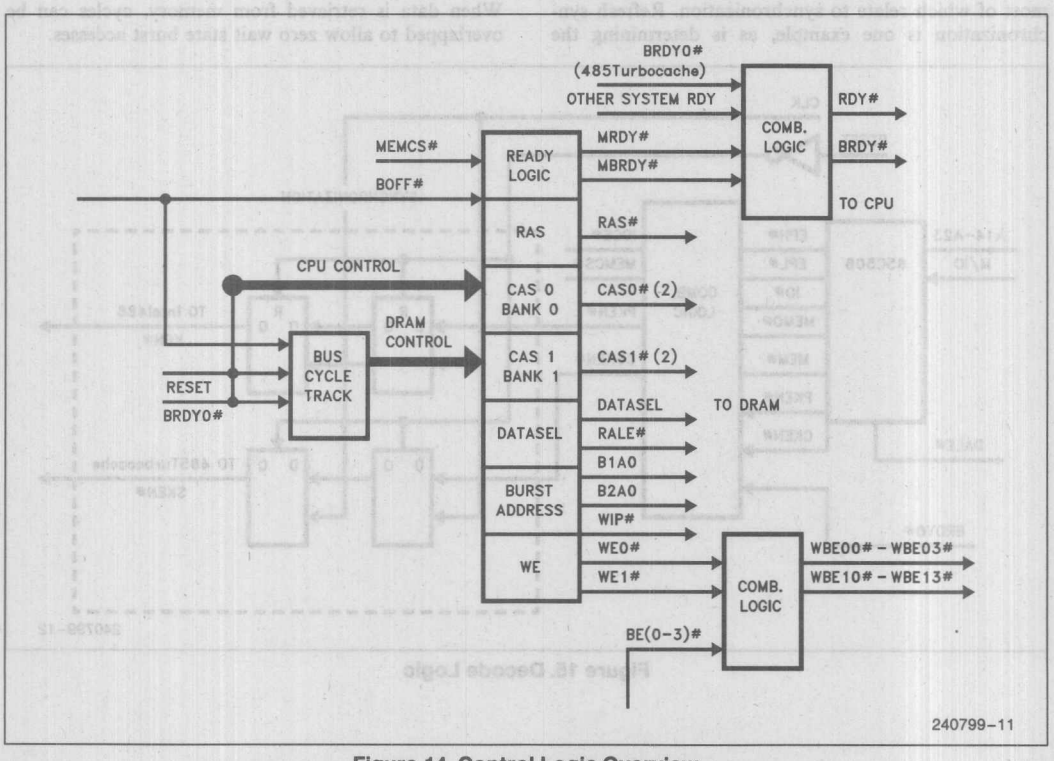


Figure 14. Control Logic Overview

Invalidation within bus cycles is another case that makes decode design difficult. The AHOLD signal must be used to implement this function. As its name implies, AHOLD can be active in any clock. If AHOLD is active in the first clock (T1) of a bus cycle, the CPU address lines are tristated in T2. Unless decode is latched at the beginning of T2, it will not be valid for the DRAM cycle.

The two-level approach allows decode to be a transparent function. The decode circuit is shown in Figure 15. The 85C508 shown here includes a flow-through latch function. Using this function, the decode outputs can be latched. The DALE# signal is generated at the beggining of the first T2 of any bus cycle. This signal activates the latch input of the 85C508. In this manner, decode is held during T2. If AHOLD is active in T1, the decode outputs may not be valid in T2. In this case, the cycle must not be started until the CPU address is redriven. Cycle-tracking PLD handles this function. By delaying the cycle start signal, the DRAM cycle is delayed. When AHOLD is deasserted, the CPU redrives the address again. At that time, CIP# is activated and the cycle begins. If AHOLD is active in any other clock, the bus cycle can continue normally.

The first level of interface with the memory subsystem, the cycle tracking PLD handles many other functions, most of which relate to synchronization. Refresh synchronization is one example, as is determining the RAS# precharge duration. CIP# is not the only signal which supports the AHOLD function. Address registers, controlled by the PLD, generate the ALD signal to disable the registers during bus cycles. These and other functions of the control logic are described completely in Section 5.11.

The PLDs in the next level of logic perform more specific functions. RAS# and CAS# are generated at this level, and the PLDs that generate these signals are devoted solely to this function. The RAS# PLD generates four RAS# signals, RAS0#–RAS3#. These signals are identical but drive different DRAM modules to reduce the load on the RAS# signal.

The RAS# function is designed to support page or static column mode memory devices. To support these devices, RAS# must be left active between accesses to the same row. The RAS# state machine is designed so that RAS is deactivated only for a refresh or page miss cycle. This module generates RAS# for both DRAM banks.

For the CAS# function, the PLD's are responsible for implementing burst accesses. During write cycles, the CAS# signals determine which DRAM bank is written to. All even doublewords (A2 = 0) are stored in bank 0 while odd doublewords (A2 = 1) are stored in bank 1. When data is retrieved from memory, cycles can be overlapped to allow zero wait state burst accesses.

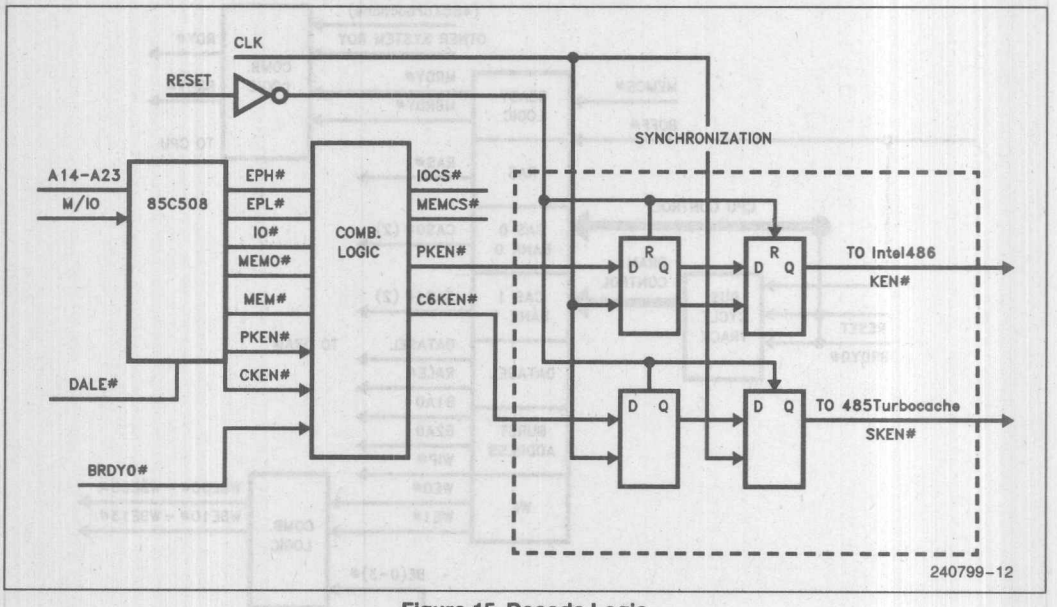


Figure 15. Decode Logic

Address generation is another important consideration in burst accesses. The address for the last three accesses of a burst must be generated by logic because the CPU cannot generate these addresses in time to allow zero-wait state accesses. The burst address logic shown in Figure 14 is actually two PLDs which generate the burst address for bank 0 and bank 1, respectively. The burst address consists of two signals—the lowest order DRAM addresses from each PLD.

Because of timing constraints, these signals are connected directly to the DRAM devices. The burst address PLD must generate the burst address, provide the multiplexor function for row and column addresses and generate the write address. The burst address signals must, therefore, reflect the value of A13 during miss cycles. During burst read and write cycles, these signals reflect A3.

B00MA0 and B01MA0 are the burst address signals for bank 0. Two identical signals are used to divide loading. B10MA0 and B11MA0 are the burst address signals for bank 1. A detailed description of the burst address function is given in Sections 5.6 and 5.16.

The DSEL PLD main function is to generate the data select signal. As described above, this signal is used during a burst to switch the data path multiplexer. It reflects the value of A2 during burst read cycles only and is one component of the burst address. The DSEL PLD also generates the RALE# signal to control the row address register described above.

BRDY# terminates all read cycles. MBRDY# is generated by the MRDY PLD and is separated from the RDY# signal to facilitate posted writes by preventing data bus contention. When a write cycle is immediately followed by a read, the read cycle must be delayed. This delay is implemented by delaying MBRDY# until the previous write cycle is complete. MBRDY# is combined with other burst ready inputs using combinatorial logic.

WIP# (write in progress) indicates to the MRDY PLD that a write is taking place, and MBRDY# is not generated unless this signal is inactive. WIP# tracks the state of the CAS# state machines.

The WE PLD generates WIP# and other signals associated with the write function. The MEN# signals control the address multiplexors and activate the write address path during write cycles. The WE# signals are used to create the DRAM W inputs and to implement byte steering. They are combined with latched CPU byte enables using combinatorial logic. In this way, DRAM W inputs are not active for unselected bytes. Data bus contention on unselected bytes is prevented by controlling the write data register output enables.

By implementing byte steering in this way the CAS# logic is simplified. The CAS# timing path is critical during burst read cycles, and by placing the byte steering logic in the write enable path, CAS# timing restrictions are eased.

The MRDY# signal terminates all write cycles. The logic used to generate this signal is unusual because it uses the ADS# input and is therefore at the first level. This configuration is needed to implement zero wait state write cycles.

MRDY# must be active by the end of the first T2 to terminate a write cycle and maintain zero wait-state performance. To meet this restriction, it must be active during any write cycle, or before decode is available because the CPU RDY# signal must not be activated during non-memory write cycles, MRDY# is inhibited by the decode output, MEMCS#, in combinatorial logic.

2

5.0 MEMORY SUBSYSTEM FUNCTION

In this section we will explore the function of the memory subsystem in detail. Each of the signals will be described, and bus cycles will be illustrated to show the memory logic function.

The bus cycle description in this section is specific to this example. Signals such as KEN# and RDY#, for example, are shown as they are driven by this particular control logic. The signals are not restricted to the timing shown here.

A list of the memory control signals follows.

Memory Interface Signals

5.1 CPU Interface Signals

KEN#	KEN# is an input to the processor, indicating whether the next bus cycle is cacheable or not. This signal is a logical AND of SKEN# and CKEN# signals.
PBRDY#	PBRDY# is the burst ready input to the processor. This is a logical AND of the BRDY# signal from the system and the BRDYO# from the second level cache.

5.2 Data Path Control

DATASEL — DATASEL reflects the value of A2 during burst accesses. It is used to control the data multiplexor for bank 0 and bank 1 data paths.

MRDY# — MRDY# enables the write data registers that are used to support write posting and terminates memory write cycles.

MBRDY# — MBRDY# is used for read cycles and enables the output of the data path multiplexor.

WE0#/WE1# — WE0# and WE1# signals enable the outputs of data write registers used for write posting. Both the signals are active during a write and CAS# determines the correct bank to which the data is written.

WBE00#-WBE03# — WBE00#–WBE03# are a combination of write enable and byte enable signals. They control which byte is written into bank 0 during a write cycle.

WBE10#-WBE13# — WBE10#–WBE13# control which byte is written into bank 1 during write cycles.

5.3 Address Path Control

ALD — ALD disables the clock input to the registers that hold the row and column addresses corresponding to the current bus cycle.

MEN0#,1# — MEN0#, MEN1# control signals are inputs to the address multiplexors and are used in selecting the read or write paths to the respective banks.

RALE# — RALE# enables the row address latch, allowing a new row address to be latched for successive bus cycles.

DALE# — DALE# activates the latch inputs of the decode logic in the first T2 of a bus cycle and holds the decode during the bus cycle.

B00MA0/B01MA0 — B00MA0 and B01MA0 are the burst address signals for bank 0. They correspond to the value of A3 during burst read cycles.

B10MA0/B11MA0 — B10MA0 and B11MA0 are the burst address signals for bank 1. They correspond to the value of A3 during burst read cycles.

5.4 DRAM Interface

HIT# — HIT# is active if the row address for the current memory cycle is the same as the previous memory cycle.

WIP# — WIP# indicates that a write cycle is in progress and a read to the DRAM needs to be delayed till WIP# becomes inactive.

CIP# — CIP# indicates a memory cycle is in progress. If the current cycle is not to DRAM, CIP# is deactivated else it remains active till the end of the bus cycle.

RAS0–3# — RAS0–3# go active for a valid row address. It remains active between accesses to the same row and is de-activated only for page miss and refresh cycles.

DRAS# — DRAS# is the delayed RAS# signal to accomodate the RAS# hold time requirements.

RFRQ — RFRQ indicates that a refresh of the DRAM is required. This signal is activated every 15.6 us.

RFACK — RFACK is asserted as a response to RFRQ and indicates that the DRAM controller is ready to perform the refresh cycle. It is active during idle cycles or after the current cycle is complete.

PCHG — PCHG determines the timing of refresh cycles and RAS# precharge count.

CAS0#/CAS1# — CAS0# and CAS1# signals are active when a valid column address is present on the bus and control the bank to which the data is written into.

MEMCS# — MEMCS# is active when a read or a write is performed to the DRAM. It is the synchronized output of the address decoder.

5.5 Controller Signals

CT — CT indicates that a new cycle had started while a cycle was in progress or the refresh cycle was taking place. It is de-activated when the pending cycle is recognized.

SKEN# — SKEN# It is an input to the second level cache and is similar to the KEN# signal input to the processor.

CKEN# — CKEN# is the output of the second level cache. It is activated twice for a valid line fill - first to enable a 485Turbocache cache line fill and the second time to validate it.

LA2, LA313 — LA2 and LA313 are latched versions of address lines A2 and A13. LA313 is the lowest order DRAM address line. The multiplexor output reflects A3 when RAS# is low and A13 when RAS# is high.

M# — M# indicates the occurrance of a write miss.

BRDYO# — BRDYO# is a burst ready signal driven by the second level cache. It is activated when a read hit occurs in this cache.

5.6 Read Cycles

Timing Diagram 16 shows a burst read cycle. At the start of the bus cycle, RAS# is inactive. This case is a rare occurence because RAS# is normally active. Unless a cycle is the first bus cycle after a reset or refresh cycle, RAS# will be active in T1.

It is useful to examine this case because it demonstrates a complete DRAM cycle. The basic function of most of the control logic is illustrated.

The cycle begins with the activation of ADS#. The controller samples this signal and activates both ALD and CIP#. The CPU address registers are disabled by ALD. Therefore, the previously latched address is held throughout the bus cycle. The latched address is valid in the first T2 of the bus cycle.

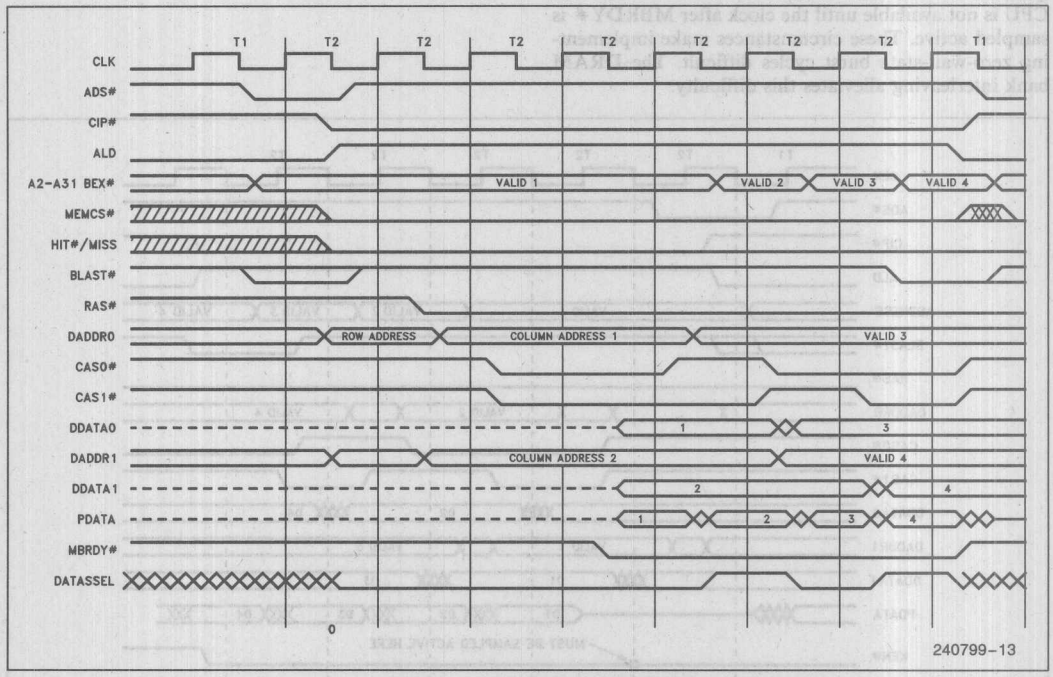


Figure 16. Burst Read Cycle

The row address comparison is made with this address. As a result, the HIT# signal is not valid until the rising edge of the second T2. At this rising clock edge, the CIP#, MEMCS# and HIT# signals are sampled. If MEMCS# is sampled active, the RAS# signal is activated.

The delay line holds the DRAS# signal high for 20 ns after RAS# is activated. In this way the row address is maintained to meet tRAH, the row address hold time. When DRAS# is activated, the address multiplexers switch to the column address path. The MEN# signals are not active, and the read path is selected.

In the third T2 of the bus cycle CAS# is asserted. This cycle begins with A2 low and the first access is to bank 0. Due to the access time of the DRAM two clocks are required to retrieve data from memory. MBRDY# is asserted in the fourth T2 of the bus cycle, and this action completes the first access of the burst read. The access is completed in five clocks. The minimum time for this access is two clocks indicating that three wait-states were added to the first cycle.

The timing diagram reveals two important points about burst cycle implementation. First DRAM access requires two clocks. Second, the burst address from the CPU is not available until the clock after MBRDY# is sampled active. These circumstances make implementing zero-wait-state burst cycles difficult. The DRAM bank interleaving alleviates this difficulty.

The first advantage of interleaving is revealed in the second and third T2 states. Access to both the first and second memory doublewords can be made simultaneously. This function requires that the burst address be predicted. As mentioned above, the burst address from the CPU is not available until several clocks later. The burst address for both the first and second accesses is generated in the second T2. Therefore, CAS# for both banks can be asserted in the next T2 state.

The second advantage of interleaving is seen in fifth T2 of the burst cycles in which DATASEL switches the data multiplexer. The second doubleword is driven on the CPU data bus. In this CLK, the burst address for the third access of the cycle is generated. CAS00# and CAS01# are also deasserted to begin the third access. Note that this access is started before the second access is completed. The cycle overlap shown allows new data to be driven on the CPU data bus every clock. This way zero-wait-state access is achieved.

Timing is even more critical during page hit cycles. Fig. 17 shows the timing of this cycle. Because of the function of RAS#, this cycle is more common than the cycle discussed above. The row address is the same as in the previous cycle. Therefore, the RAS# signal is left active.

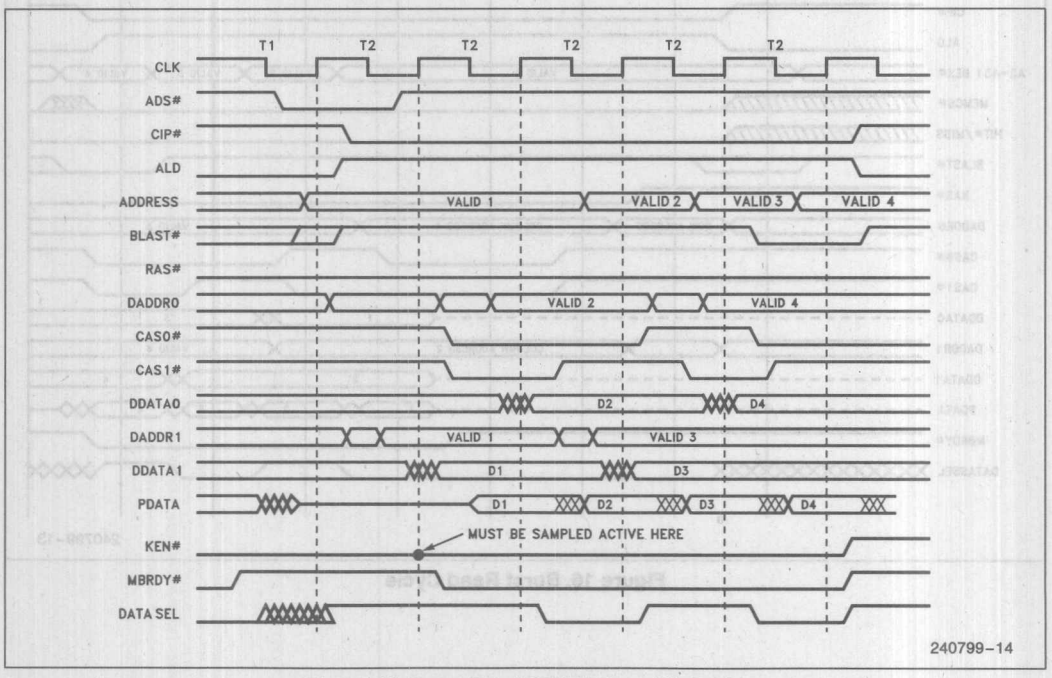


Figure 17. Burst Read DRAM Page Hit Cycle

When a burst read starts with RAS# active, fewer clocks are required to complete the first access. This reduction improves performance. As a result, however, some timings become more critical. One of these is the time allowed to generate the burst address.

The CAS# signals are asserted in the second T2 of the bus cycle. MBRDY# is also asserted at this time. To meet the address access time of the DRAMS, the burst address must be generated in the second T2. The rest of the read column address must also be available at this time. Two logic functions are needed to meet this timing requirement. First, read and write address paths must be separate to allow the read address to be available in the first T2. Second, the burst address path logic must latch the CPU A3 signal directly. In this way, the logic can generate the necessary address in time. The burst address state machine must track the state of A3 at the begining of every cycle. The state machine function is described in Section 5.11.

The timing of KEN# must also be considered in this example. KEN# must be valid at the beginning of the second T2 of the cycle. If it is not, the cycle will not be cached, and a 16-byte access cannot be generated. If KEN# is active, a 16-byte burst access will be generated, and the cycle will be cached as long as KEN# is active in the second to last T2.

At first glance this timing may not appear critical. KEN# is a decode function, and decode is valid at the clock edge called for. The KEN# input to the CPU must be synchronized to the CPU clock, however. Since decode is not synchronous, a two-clock synchronizer delay is required, and this delay is the reason that KEN# is normally active in this example.

From the time CAS# is activated, this cycle is exactly the same as in the previously described burst cycle. It is terminated when BLAST# is asserted, and MBRDY# is deasserted when BLAST# is sampled active.

5.7 Write Cycles

As described in Section 4.1, a posted or delayed write function is employed in this example to reduce write cycle latency. Latency is reduced since write cyles are overlapped with other cycles including other Write cycles or reads from the second level cache. Write cycles normally make up 70 percent of all cycles, and overlapping can increase performance accordingly.

Figure 18 illustrates the posted write implementation. In this example cycles begin when RAS# is inactive. As with read cycles, this case is rare in practice.

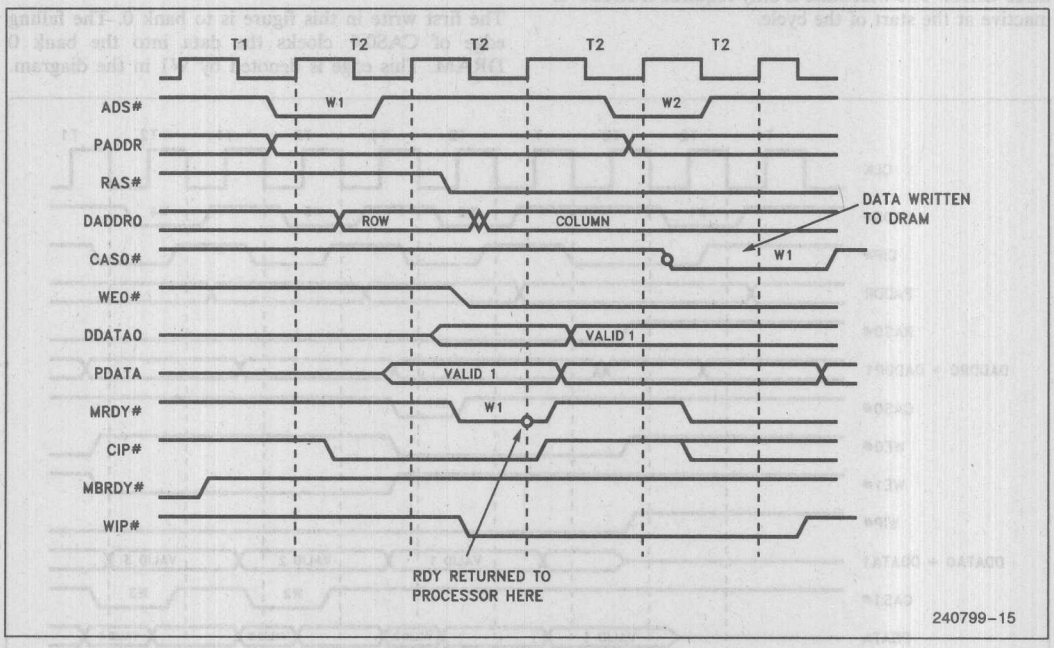


Figure 18. Basic Write Cycle

The cycle begins like a read. The CPU drives ADS# active, and the decode is sampled. RAS# is activated if the cycle is in DRAM space. In the second T2 of the cycle, however, the latched version of W/R# (LW/R#) is sampled active at the rising edge of the second T2. In response, the control logic begins several write cycle functions at this clock edge.

The CAS# state machine for the appropriate bank enters the write sequence. The MEN# and WE# signals are asserted. MRDY# is also asserted, terminating the cycle at the CPU. The MEN# signals activate the write address path. This address is not present at the multiplexor outputs, however, until the next clock at which the write pipeline register latches the write address.

The write data is latched at the same clock edge. The write data registers are enabled by MRDY# which simultaneously terminates the CPU cycle. Note that data is latched in both the bank 0 and bank 1 registers.

The WE0# and WE1# signals are also both active. The CAS# signals determine which bank is written to. These signals are asserted within two clocks after MRDY#. This action completes the write cycle. Note that, while five clocks are required clocks are required to complete the cycle, the CPU cycle is terminated in three CLKs. The wait state is only required if RAS# is inactive at the start of the cycle.

In Figure 18 the next bus cycle starts immediately after RDY# is sampled. In this case, CAS# is activated during the second clock of the next bus cycle. This overlap of cycles is similar to the pipelining feature used by many processors except that the Intel486 processor bus is not involved in the posting function. All logic for this function is implemented in the memory controller.

Figure 19 is a more typical Intel486 processor bus sequence which clearly illustrates the advantages of the posting technique. Four write cycles have occurred together without idle bus clocks occurring between cycles. Since all writes access the same DRAM row, RAS# is active throughout the sequence.

Without the extra clock to activate RAS#, MRDY# can be asserted in the clock after ADS# is asserted. These cycles, therefore, have no wait-states. As before, the write cycle is not complete when MRDY# is asserted but instead when CAS# is asserted two clocks after MRDY# to terminate the CPU bus cycle.

At zero wait-states, each write cycle still requires four clock cycles. The last two clocks of each write cycle overlap with the next cycle. The net effect on the CPU bus is the same as a string of two-clock write cycles, as illustrated in Figure 19.

The first write in this figure is to bank 0. The falling edge of CAS0# clocks the data into the bank 0 DRAM. This edge is denoted by W1 in the diagram.

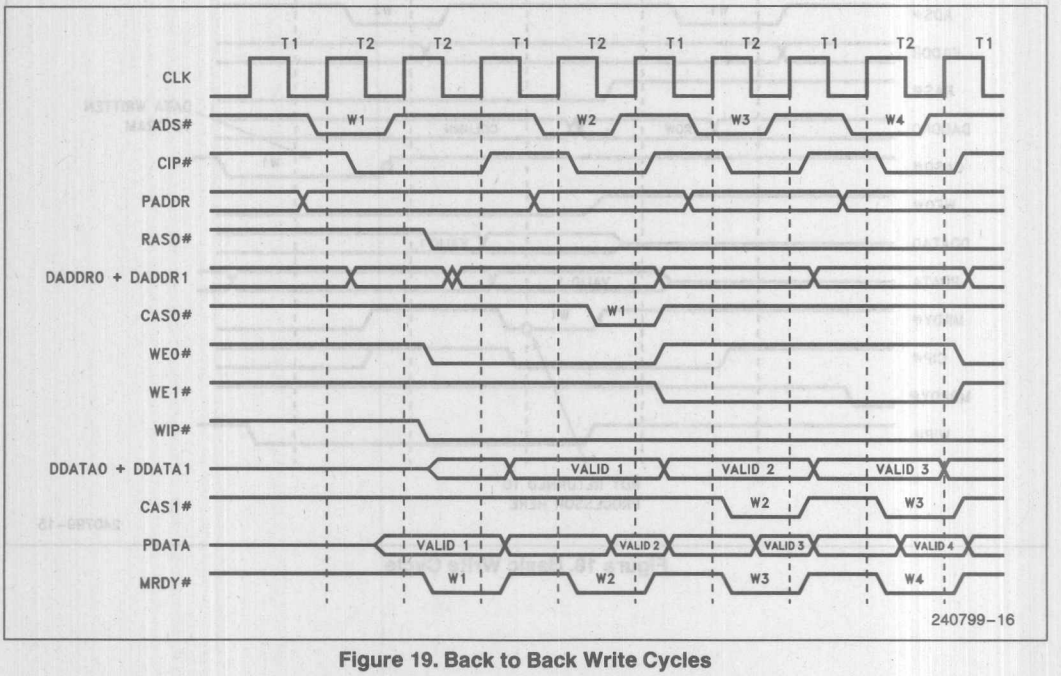


Figure 19. Back to Back Write Cycles

CAS0# is asserted in the same clock that MRDY# terminates the second write (W2), which accesses bank 1. CAS1# is activated in the same clock as MRDY# for the third write (W3).

The second and third writes happen to be to the same DRAM bank. As we see, no timing modification is required in this case. Write cycles can be completed with zero wait states in either case. This is important since writes often occur in sequence on the Intel486 bus, but not necessarily to sequential addresses. Write posting supports zero wait-state write cycles to sequential and non-sequential addresses.

This fact is also important if the design is to be modified. For example while, interleaved DRAMs may not be required in systems with a permanent second level-cache, the write posting technique may still be used in the system. The benefits of this technique still apply since write cycles may still be overlapped as described.

5.8 Consecutive Bus Cycles

The DRAM control logic is optimized for write cycles, as warranted by the Intel486 processor's bus characteristics. Over 70 percent of all cycles are writes. By employing the posted write technique, system performance is increased.

The posted write technique poses some special problems, however. Page miss, refresh and consecutive write-read cycles require special consideration. We will begin by discussing the consecutive write-read case. Page miss and refresh cycles will be discussed in sections 5.9 and 5.10.

When a read cycle immediately follows a write, the read cycle must be delayed as illustrated in Figure 20. The read cycle is delayed to allow the write to complete. Only read cycles to DRAM, i.e. (cache misses) need be delayed. Cache hits and write cycles overlap easily because the cache is on the CPU side of the DRAM controller.

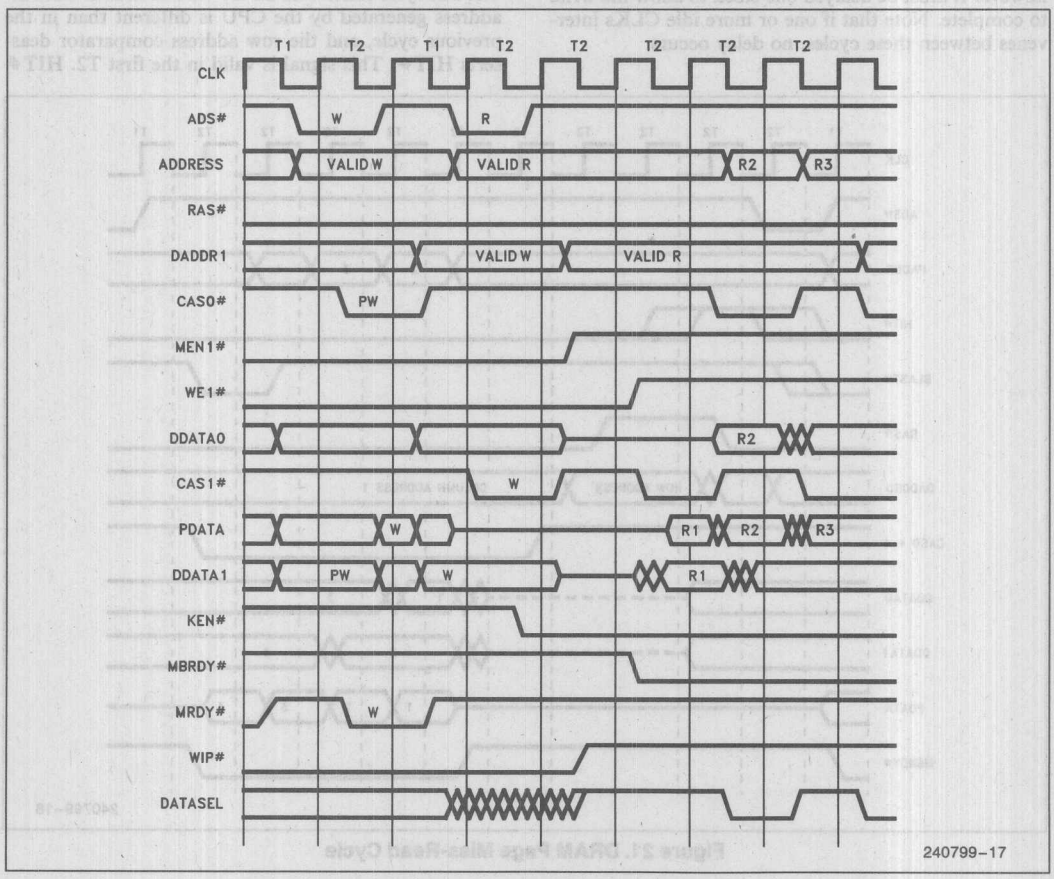


Figure 20. Consecutive Write-Read Cycle

Write cycles cannot overlap DRAM read cycles, however, primarily because of data bus contention. The DRAMs used here have common data I/O pins. In this case read and write data paths cannot be active at the same time.

To prevent data bus contention, the first data access of the read is delayed. In Figure 20 the first read access is to the same bank as the write. In addition, the read cycle accesses the same DRAM row. Two functions are required to ensure that the write is completed. First, the write address must be held until CAS# is asserted. Second, the data mux outputs must not be enabled until the CPU tristates the bus.

The first function is accomplished by the MEN# signals. The MEN# state machine tracks the CAS# function for the appropriate bank. When the write for that bank is complete, MEN# is deactivated. In this way, the read address path is not enabled until the CLK after CAS# becomes active. Normally, the read address would be valid in the first T2 of the read cycle; however it must be delayed one clock to allow the write to complete. Note that if one or more idle CLKs intervenes between these cycles, no delay occurs.

The second function is accomplished with the WIP# signal which is active until all write cycles are complete. A read cycle to either bank will be delayed if it immediately follows a write. The first access of the read is delayed by MBRDY#, which is not asserted until the WIP# signal is deasserted. WIP# is deasserted once all pending writes are complete.

5.9 Page Miss Cycles

As described previously, page miss cycles occur when the CPU generates a cycle which changes the DRAM row address. The RAS# signal must be deasserted to change the ROW address in the DRAMS. Any time RAS# is deasserted, it must remain high for the precharge time (t_{RP}). A delay is added to every page miss cycle to satisfy this requirement.

For read cycles this function simply requires extra wait states as illustrated in Figure 21.

The bus cycle starts with RAS# low or active. The row address generated by the CPU is different than in the previous cycle, and the row address comparator deasserts HIT#. This signal is valid in the first T2. HIT#

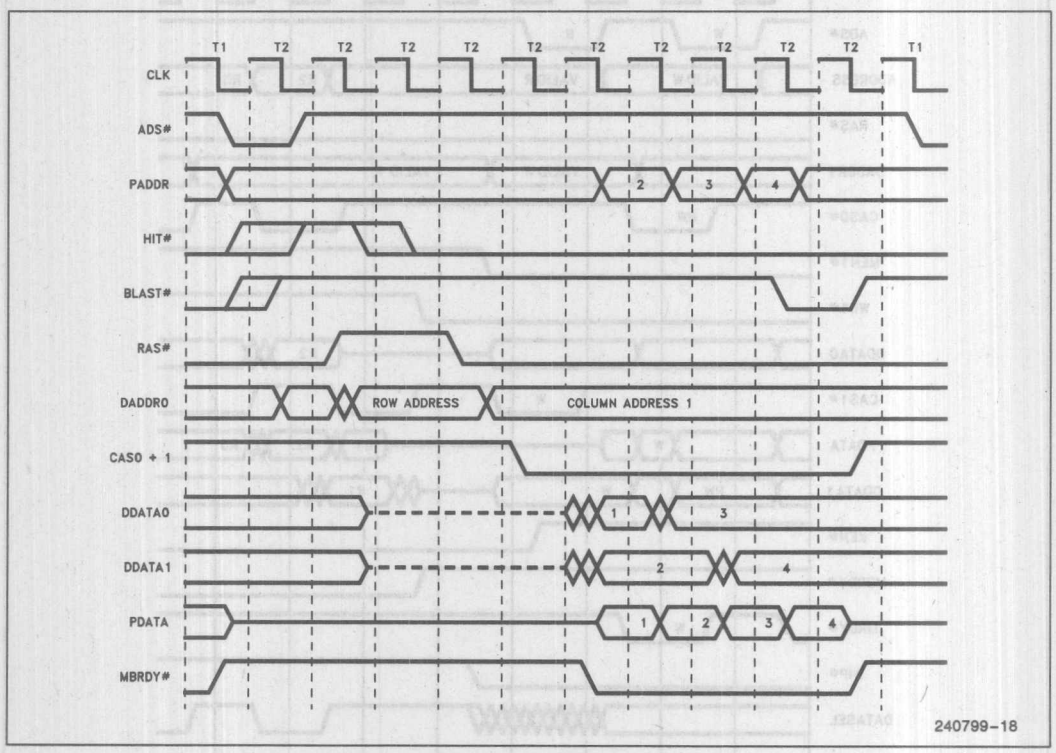


Figure 21. DRAM Page Miss-Read Cycle

is sampled by the RAS# PLD at the rising edge of the second T2. In response, RAS# is immediately deasserted and held inactive for two clocks. This time satisfies the RAS# precharge requirement.

Four wait states are added to process the miss cycle. These clocks are added to every read cycle which accesses a new DRAM row. The delay is accomplished, again, with the MBRDY# signal. MBRDY# will not be asserted when RAS# is inactive. Once RAS# is sampled active, MBRDY# is asserted. From here, the cycle proceeds as described in section 5.6.

Write miss cycles are more complex than read miss cycles, due mainly to the write posting technique. The added complexity results in lower latency than in a non-posted memory system, however. Figure 22 illustrates how this improvement is achieved.

The write cycle in Figure 22 also begins with RAS# active. The HIT# signal is deasserted in the first T2 at the same time that MRDY# is asserted. MRDY# could be inhibited at this point to prevent write cycle termination. The wait-states added to meet RAS# precharge time would then be added to this cycle. Five wait states are required to meet the precharge time.

The average number of write cycle clocks can be reduced, however, if another method is used. MRDY# can be allowed to terminate the cycle. In this case, any necessary wait-states will be added to the next cycle.

This method improves the average in two ways. First, some write miss cycles will not require wait-states. This is the case when the next cycle occurs four or more clocks after a write miss. In addition, wait states will be reduced when the next cycle occurs in two or three clocks. Second, three wait-states are required to complete the next cycle when it follows immediately as illustrated in Figure 22.

The first cycle in this figure is a page miss. It is terminated at the CPU without wait-states. Because HIT# is not active in the first T2, RAS# is deasserted. At this point, additional clocks are added to perform the miss

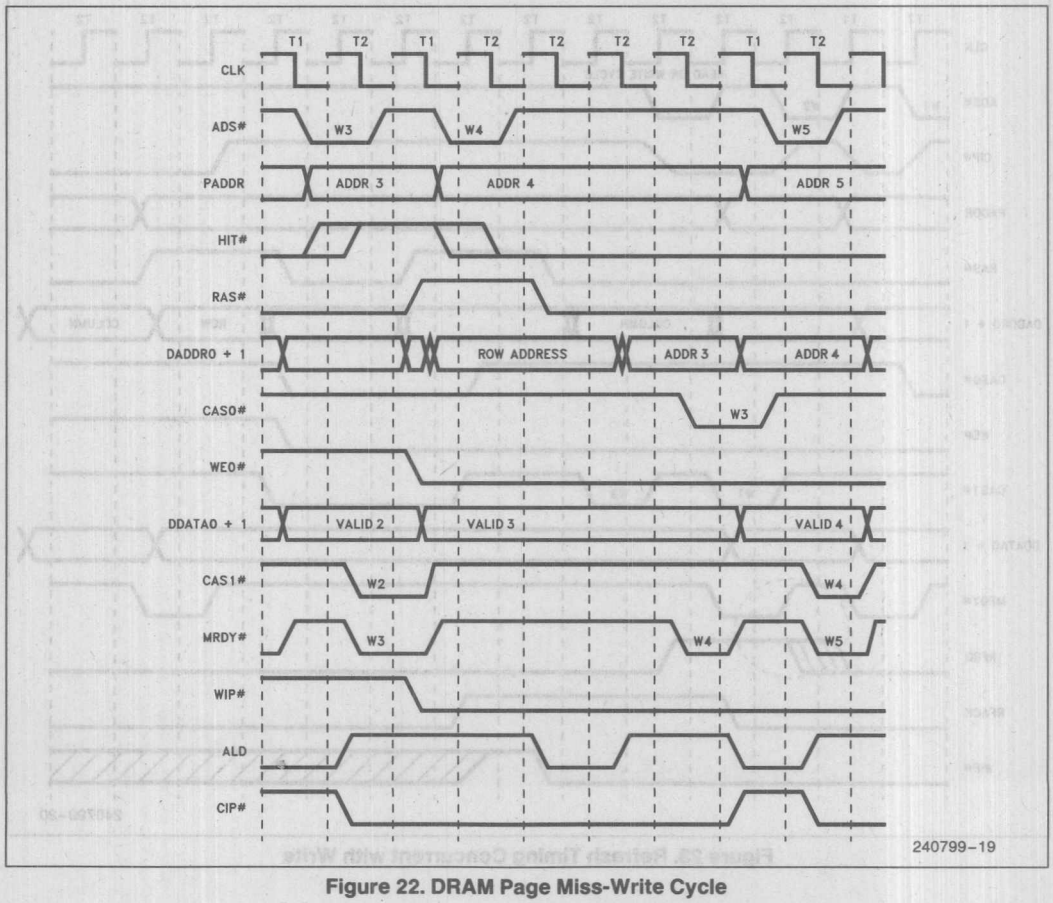


Figure 22. DRAM Page Miss-Write Cycle

function. Part of the time required for RAS# precharge is overlapped with the next cycle. The two clock overlap reduces the number of wait-states required in the next cycle. Therefore, the average write cycle latency is reduced.

5.10 Refresh Cycles

The CAS# before RAS# refresh function is used in this example. This function uses internal counters in the DRAM devices to generate the refresh address. When the CAS# input is activated prior to RAS#, the internal counter is incremented. The output of the counter is then used as the address of the row to be refreshed.

Each refresh cycle refreshes one row of the DRAM array. The refresh cycles are distributed such that one occurs every 15.6 μs, with every row being refreshed in 8 ms. Refresh cycles are initiated by the RFRQ signal. This signal is activated every 15.6 μs by a counter.

RFACK is asserted in response to RFRQ. This signal indicates that the DRAM controller is ready to perform the refresh cycle. It also signals the counter circuit that RFRQ can be deasserted.

The function of RFRQ and RFACK is very similar to that of the CPU's HOLD and HLDA signals. RFRQ is sampled at the end of each cycle and during idle cycles. RFACK is activated in the clock after RFRQ is sampled, except immediately after write cycles.

Again, the posted write function must complete before the refresh cycle begins. If WIP# is active when RFRQ is sampled, RFACK will not be immediately asserted. RFACK will be asserted after WIP# is deactivated as illustrated in Figure 23.

Another cycle can start between RFRQ and RFACK. The cycle start PLD tracks this case. CIP# will not be asserted for any cycle that starts during this interval. Once the refresh cycle is complete, this cycle can be started.

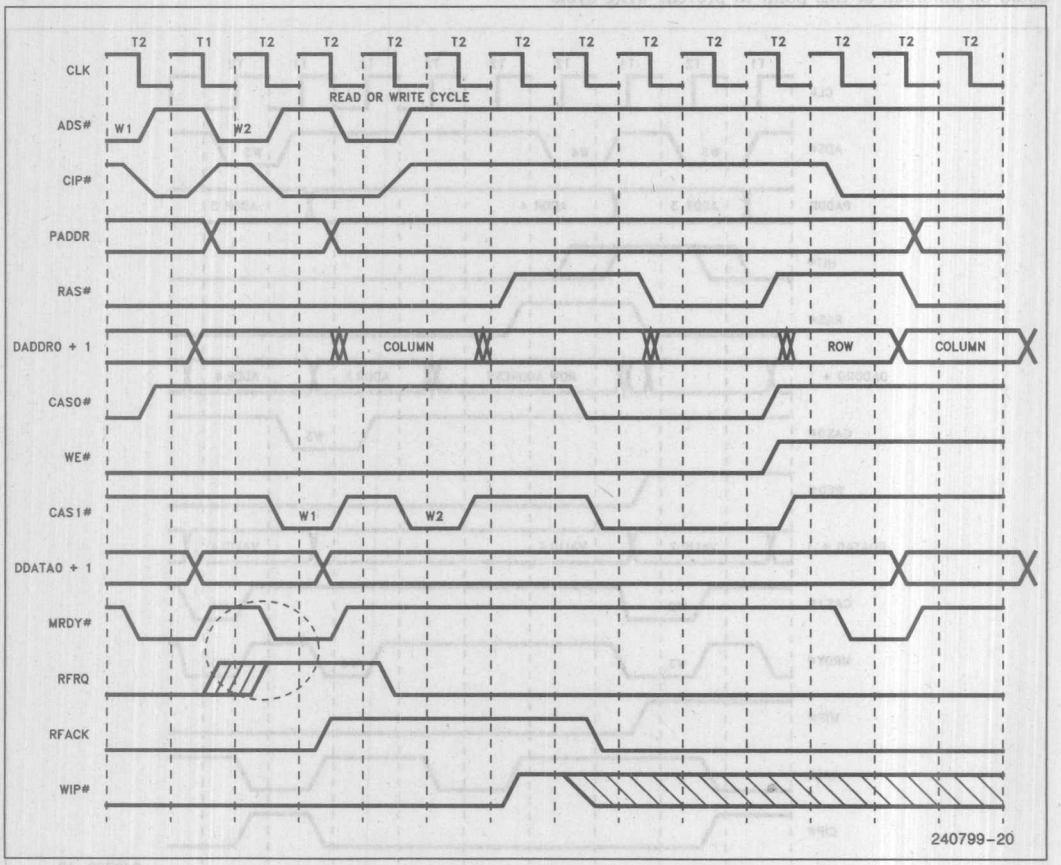


Figure 23. Refresh Timing Concurrent with Write

6.0 CONTROLLER IMPLEMENTATION

The functions described in the previous section are generated by the control logic. The controller, as outlined in Section 4.0, is made up of several PLDs. These devices generate the control signals described in Section 5.0. The function of the logic is determined by the state machine definition. These state machines are distributed in the different PLDs of the controller.

In this section, we will explore the implementation of the control logic. The discussion will focus on the state machine definition. Certain conventions are followed throughout the discussion. These conventions are based on the state machine compiler used to generate the PLD equations. This compiler uses the exclamation point (!) to indicate the low or "0" condition of a signal. It uses the number symbols (#) to indicate that the signal is active low. For example, !ADS# indicates that the ADS signal is both low and active. The # symbol indicates that a signal is active when low. So symbol !ALD means that the ALD signal is not active. These symbols are used to indicate state transitions as shown in Figure 24. The state transition in Figure 24 depends on three signals: ADS#, ALD, and RAS#. The equation indicates that if both ADS# and ALD are active or if RAS# is not active at the next clock edge. the transition from S0 to S1 takes place. In the transition between S0 and S1, the Y# signal is activated. The definition of states indicates which outputs are changed in the transition. These conventions are used to describe the control state machines in the next section.

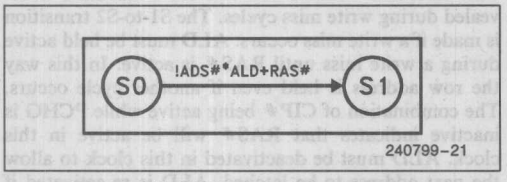


Figure 24. State Transition Example

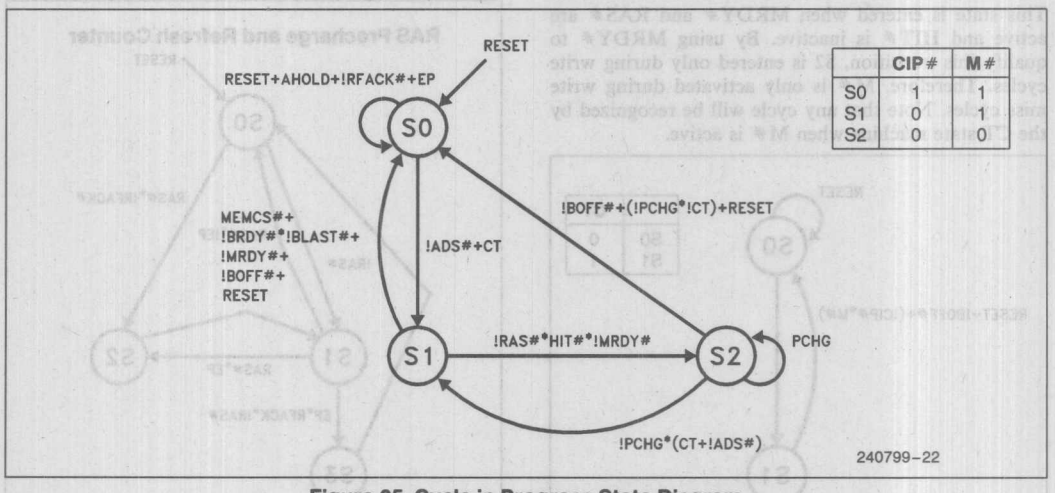
6.1 Cycle Tracking Logic

The cycle tracking logic is contained in one PLD. The five state machines implemented in this PLD start and end DRAM cycles, control refresh timing and control the address registers. These state machines, along with the MRDY# state machine comprise the first level of control logic. All other control state machines depend on this first level to generate signals at the proper time.

The signals generated by this PLD are the following:

CIP# - Cycle in Progress
ALD - Address Latch Disable
CT - Cycle Track
RFACK - Refresh Acknowledge
PCHG - RAS Precharge Count
M# - Write Miss Indicator

The primary cycle tracking state machine is shown in Figure 25. This state machine generates the CIP# and M# signals. CIP# indicates that the CPU has started a



	CIP#	M#
S0	1	1
S1	0	1
S2	0	0

Figure 25. Cycle in Progress State Diagram

cycle. When it is active, the rest of the logic samples the CPU control and MEMCS# signals. If the current cycle is not to DRAM, it will be ignored and CIP# will be deactivated.

This function is defined by the S0 and S1 states in Figure 25. As shown, CIP# is activated when either ADS# or CT are sampled active. If the cycle is not to a DRAM address, the MEMCS# signal will not be active in the next clock. In this case, CIP# is deactivated to wait for the next ADS#. If the cycle is to DRAM, CIP# stays active until the end of the bus cycle. The bus cycle is terminated by one of three circumstances. All write cycles are terminated with the MRDY# signal. Read cycles are terminated by BRDY# and by BLAST#. The cycle can be aborted by BOFF#. Any of these three events causes CIP# to be deactivated (S1 to S0).

Two special cases are also handled by this state machine. When AHOLD is active in the same clock as ADS#, MEMCS# is not valid. In this case, the CIP# signal is not activated until AHOLD is deasserted. The state machine remains in S0 when AHOLD is active.

The second case is a write miss cycle. During a write miss, CIP# must be active for the cycle to complete. CIP# is active in this case after MRDY# is returned to the CPU. Cycles that start during the time CIP# is active must be tracked by the CT state machine. The M# signal indicates to the CT state machine that the cycles must be tracked.

The state in which M# and CIP# are both active is S2. This state is entered when MRDY# and RAS# are active and HIT# is inactive. By using MRDY# to qualify this transition, S2 is entered only during write cycles. Therefore, M# is only activated during write miss cycles. Note that any cycle will be recognized by the CT state machine when M# is active.

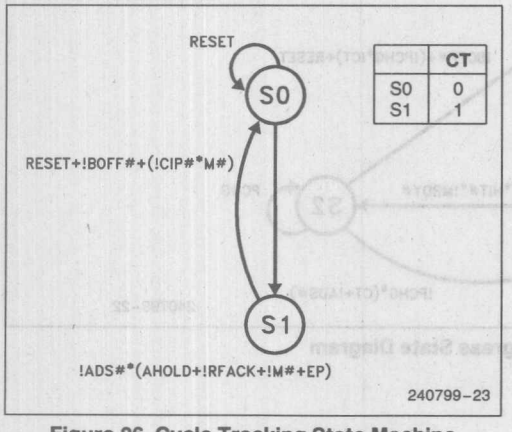


Figure 26. Cycle Tracking State Machine

The CT state machine is shown in Figure 26. This state machine tracks cycles that start while the CIP# state machine is busy. It tracks CPU cycles that start during refresh cycles as well as the two cases mentioned above.

This state machine tracks one cycle. Any cycle that starts while CIP# is busy is not terminated immediately. The MRDY# and MBRDY# signals are delayed until the previous cycle is finished. Therefore, anytime CT is active, there is only one cycle pending.

CT is deactivated when the pending cycle is recognized by the CIP# state machine. This event is indicated by CIP# active and M# inactive. When this event occurs, the CT state machine transitions to S0 deactivating CT.

The ALD signal is also active only during DRAM cycles. Therefore, its state machine is very similar to that of CIP#. As with CIP#, ALD is asserted when ADS# is sampled active. If the cycle is not to a DRAM address, ALD is deasserted. When a DRAM cycle is terminated, ALD is also deasserted. The S0- to-S1 transition is quite similar to that of CIP#.

The difference between the two state machines is revealed during write miss cycles. The S1-to-S2 transition is made if a write miss occurs. ALD must be held active during a write miss until RAS# is active. In this way the row address is held even if another cycle occurs. The combination of CIP# being active while PCHG is inactive indicates that RAS# will be active in this clock. ALD must be deactivated in this clock to allow the next address to be latched. ALD is re-activated if

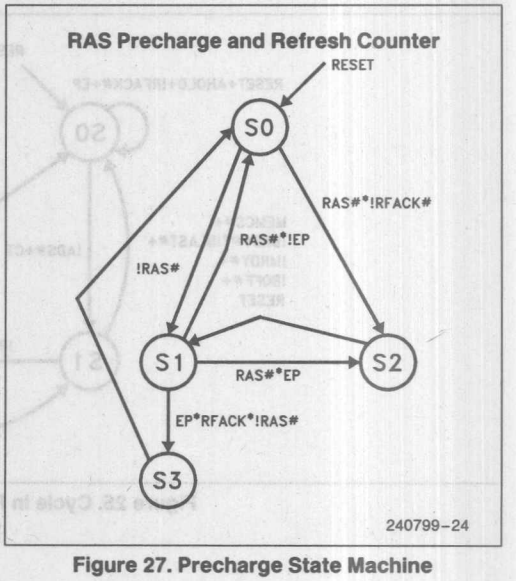


Figure 27. Precharge State Machine

another cycle has started during the write miss process. CIP# and MEMCS# are sampled during S0 for this purpose.

The PCHG state machine provides two functions. It determines the time RAS# is inactive during a miss or refresh cycle, and it determines the timing of refresh cycles. Figure 27 shows the state transitions of the PCHG state machine. Because the timing of this signal is not obvious, Figure 28 has been included. It shows a refresh cycle which occurs following a write cycle.

After RAS# is active the PCHG signal is activated. State S1 is maintained then until RAS# is deactivated. RAS# is only deactivated during a miss or refresh cycle or, of course, if RESET is asserted. During a miss cycle the transition to S0 is made deactivating PCHG. RAS# is then deactivated, resulting in two CPU clocks of RAS# precharge time.

States S1 and S2 define the timing of refresh cycles. The transition to this sequence is made when RAS# is sampled inactive while EP is active. EP indicates that the RAS# state machine has entered the refresh sequence.

RFACK initiates the refresh sequence. It indicates that the control logic is ready to accept a refresh request. The RFRQ signal is sampled at the end of a DRAM cycle or during idle clocks. Note that RFRQ cannot be recognized during a write miss.

RFACK is deactivated after RAS# is deactivated at the beginning of the refresh sequence (See Figure 27 and Figure 28).

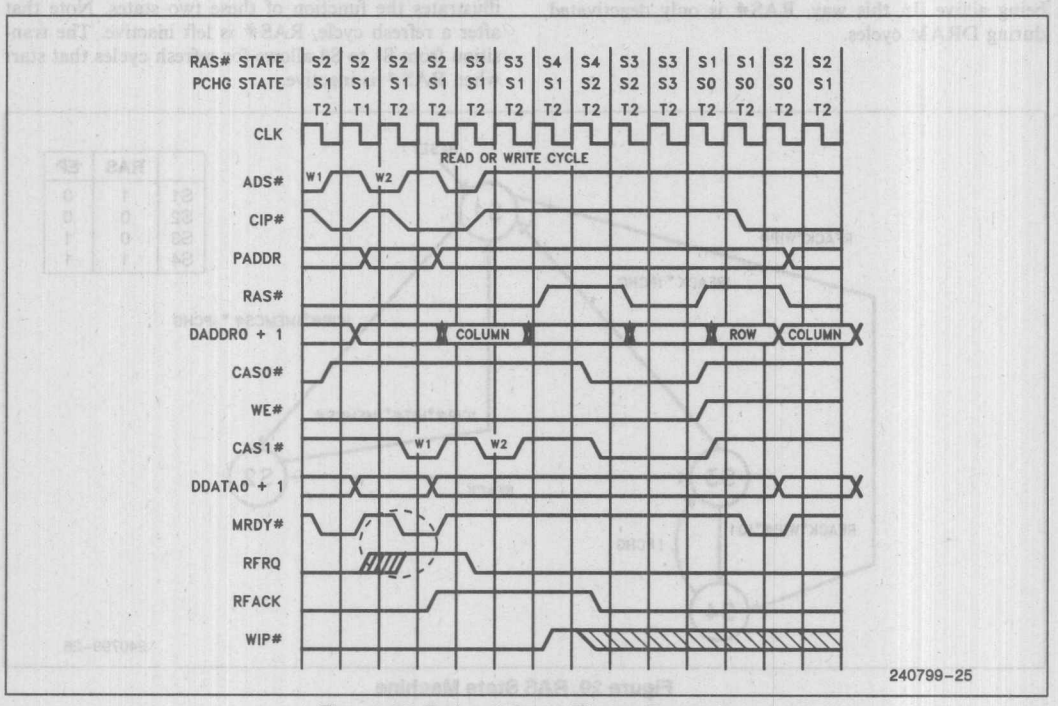


Figure 28. Refresh State-Timing Example

6.2 RAS# Logic

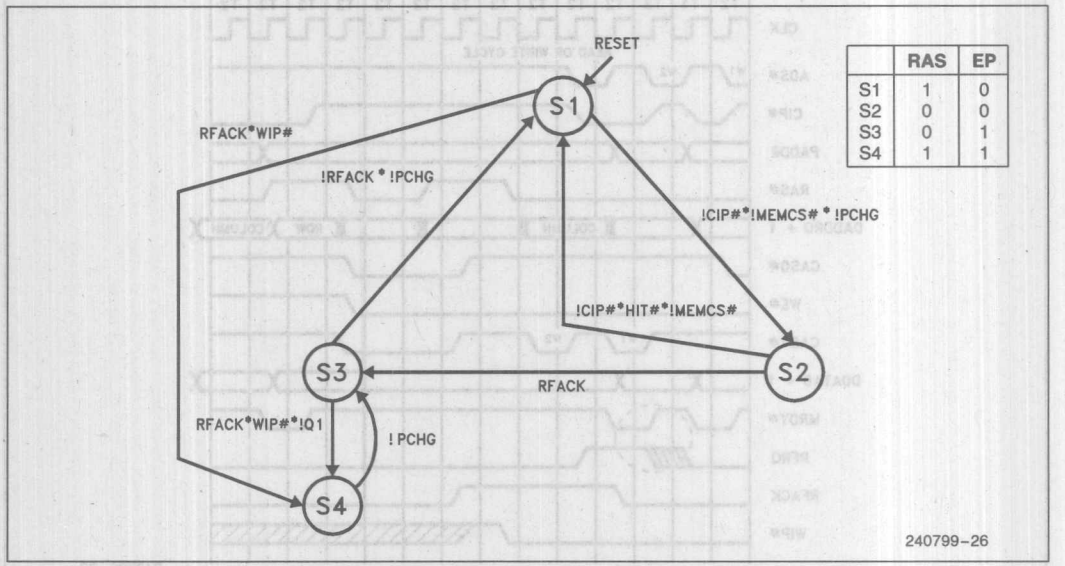
The RAS# logic for both memory banks occupies one PLD. Four RAS# signals are generated: RAS0# – RAS3#. These signals are generated to divide loading. Their timing is identical. The state machine for RAS is relatively simple and is shown in Figure 29.

States S1 and S2 are used to implement RAS# function for normal cycles. After RESET, the state machine waits for the first bus cycle. The first bus cycle is signaled by the CIP# signal. When CIP#, MEMCS# and PCHG are sampled active, RAS# is asserted. RAS# stays active until a miss or refresh cycle occurs.

A miss cycle is indicated when the HIT# signal is driven inactive. It is qualified by CIP# and MEMCS# being active. In this way, RAS# is only deactivated during DRAM cycles.

Once RAS# is deasserted during a miss cycle, it stays high until PCHG is sampled active. This function implements the RAS# precharge time. CIP# and MEMCS# will still be active during read miss cycles. Therefore, RAS# will be asserted in the next clock. For write miss cycles the WIP# signal must be used to restart RAS#. With a write miss, a non-DRAM cycle can occur before RAS# is asserted. WIP# is the only valid indication that a DRAM cycle has occurred in this case. WIP# is combined with MEMCS# to create the CSWIP# term which indicates a valid RAS# cycle.

When a refresh cycle occurs, the RAS# state machine transitions to S3. S3 and S4 are devoted to the refresh function. When RFACK is sampled active, the transition occurs. The refresh sequence shown in Figure 28 illustrates the function of these two states. Note that after a refresh cycle, RAS# is left inactive. The transition from S1 to S4 allows for refresh cycles that start when RAS# is inactive.



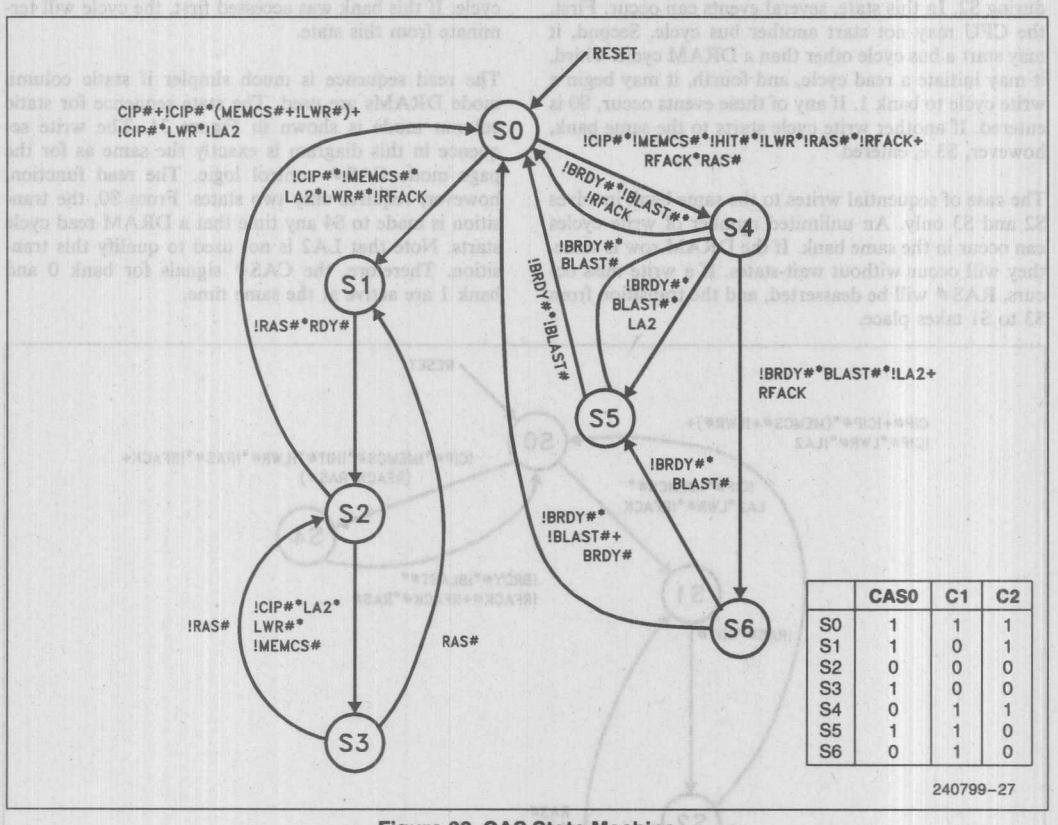
	RAS	EP
S1	1	0
S2	0	0
S3	0	1
S4	1	1

Figure 29. RAS State Machine

6.3 CAS# Logic

Two separate PLDs implement the CAS# function. These PLDs generate the CAS# signals for bank 0 and bank 1, respectively. The state machines which generate these signals are separate and independent. Each generates two CAS# signals. CAS00# and CAS01# for bank 0, and CAS10# and CAS11# for bank1. These signals drive separate DRAM modules due to drive requirements.

Figure 30 shows the state diagram for the bank 0 CAS# function. The states on the left side of the diagram implement the write function. The states on the right implement the read function. As with RAS#, the state machine waits until CIP# indicates that a cycle has started. When CIP# is active, the state of the latched version of W/R# determines which sequence is started.



	CAS0	C1	C2
S0	1	1	1
S1	1	0	1
S2	0	0	0
S3	1	0	0
S4	0	1	1
S5	1	1	0
S6	0	1	0

Figure 30. CAS State Machine

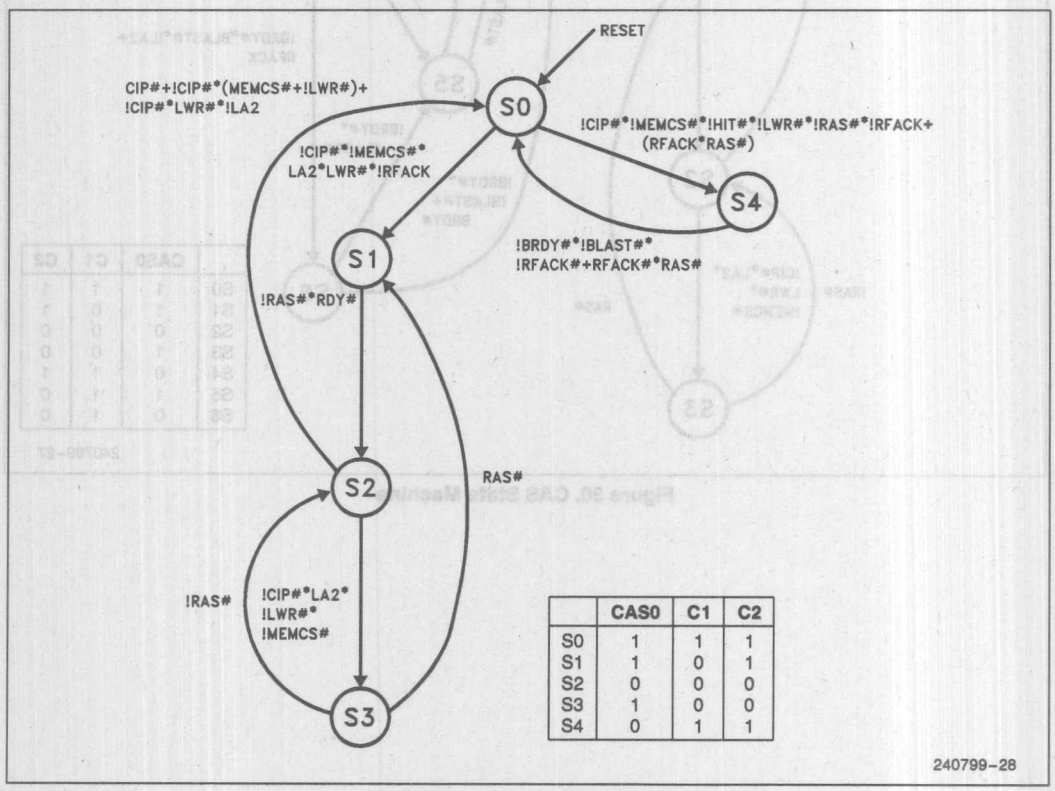
If the cycle is a read, S4 is entered. If the cycle is a write, LA2 is sampled to determine if the cycle is to bank 0. If LA2 is low, S1 is entered. Note that this function is the same for the bank 1 state machine. The only difference is the state of LA2, which starts the write sequence.

During a write cycle, CAS# is held inactive until the clock after RDY# is asserted. The state machine also waits in S1 during a write miss cycle. CAS# is asserted during S2. In this state, several events can occur. First, the CPU may not start another bus cycle. Second, it may start a bus cycle other than a DRAM cycle. Third, it may initiate a read cycle, and fourth, it may begin a write cycle to bank 1. If any of these events occur, S0 is entered. If another write cycle starts to the same bank, however, S3 is entered.

The case of sequential writes to the same bank involves S2 and S3 only. An unlimited number of write cycles can occur in the same bank. If the DRAM row is same, they will occur without wait-states. If a write miss occurs, RAS# will be deasserted, and the transition from S3 to S1 takes place.

During read cycles, the CAS# signals for bank 0 and bank 1 are activated at the same time. Therefore, the state machines enter S4 at the same clock. At this point, however, the state of LA2 determines which state machine enters S5. In S5, CAS# is deasserted to prepare that bank for the next access. If S6 is entered, the data from that bank has not yet been accessed. CAS# must be held active, in this case, until the data is sampled by the CPU. From S6, the next transition will be to S5 to continue the cycle, or S0 to terminate the cycle. If this bank was accessed first, the cycle will terminate from this state.

The read sequence is much simpler if static column mode DRAMs are used. The state sequence for static column mode is shown in Figure 31. The write sequence in this diagram is exactly the same as for the page mode CAS# control logic. The read function, however, requires only two states. From S0, the transition is made to S4 any time that a DRAM read cycle starts. Note that LA2 is not used to qualify this transition. Therefore, the CAS# signals for bank 0 and bank 1 are active at the same time.



	CAS0	C1	C2
S0	1	1	1
S1	1	0	1
S2	0	0	0
S3	1	0	0
S4	0	1	1

Figure 31. Static Column CAS State Machine

6.4 Write Control Logic

The posted write implementation requires logic support for a few key functions. These functions are required mainly to support posting with interleaved memory. Three types of signals are generated to implement these functions:

Multiplexer Select - These signals control the address multiplexers when RAS# is active. During write cycles, they must be active to select the write address path. These signals stay active during read cycles which are immediately preceded by a write. They are deactivated, when the write cycle is complete. Once they are deactivated the read cycle may proceed as the read path is selected.

Write Enable - These signals are combined with the byte enable CPU outputs (BE0# – BE3#) to create the WBE# signals. The WBE00# – WBE03# signals control which byte is written in bank 0 during a write cycle. The WBE10# – WBE13# signals perform the same function for bank 1.

Write In Progress - This signal is active when a write cycle has been started by either DRAM bank. It is active when either C01# or C11# is active. C01# and C11# are state outputs from the CAS# state machine which indicates that a write cycle is being performed. C01# is generated for bank 0 and C11# for bank 1. WIP# is only required for interleaved memory systems. The C01# (or C11#) output would be sufficient for a non-interleaved (single bank) system.

The state machines which generate these signals are shown in figure 32. The state diagram for the MEN0# signal is shown. This signal enables the address multiplexer for bank 0. MEN0# is activated whenever a write cycle occurs to an address with A2 low (0). The MEN1# function is the same except that it is activated when A2 is high (1). The CIP#, MEMCS# and LWR# signals are used to indicate a valid write cycle.

The MEN# signals are deactivated when the write cycle is complete. The cycle is complete when CAS# for that bank is sampled active. For bank 0, C01# is used to indicate that a write is in progress. MEN0# is held active when C01# is active. When CAS00# is sampled active, CIP# is checked to determine if another valid write to the same bank has occured. If so, MEN0# stays active until CAS00# is sampled active. This function keeps the write address path open during consecutive writes to the same bank.

The WE# state machine is very similar to that of the MEN# state machine. When a write cycle starts, WE0# is activated in the same manner as MEN0#. The write enable signals, however, must stay active one clock longer than the MEN# signals. Therefore, the WE# signal is not deactivated until C01# is sampled inactive.

WIP# is generated in part by combinatorial logic so that it can be active in the same clock as the C01# and C11# signals. WIP# must be active in this clock to ensure that a write miss is completed before a refresh cycle takes place. WIP# must also be held active one clock after C01# and C02# are sampled inactive. This timing ensures the proper sequence for subsequent read cycles. The logic equation and state machine for WIP# are shown in Figure 32.

6.5 Burst Address Logic

The burst address logic generates the B1MA0 and B0MA0 signals. These signals are connected directly to the low order address inputs of the DRAMs. Because of the direct connection, these signals must perform several different functions. They must multiplex the low order row and column addresses, multiplex the write and read addresses and generate the burst address during read cycles.

These functions are performed separately for each bank by two PLDs. Each PLD generates two identical signals to reduce the drive requirements. These signals are connected directly to two bytes of the DRAM array. The signals are generated partly by combinatorial logic and partly by the state machine.

The logic equations and state diagram for this function are shown in Figure 33. The state machine generates the burst address for read cycles. The logic equations handle the multiplexing functions.

The burst address is generated after a burst read cycle has started. Note that the Intel486 CPU cache need not be enabled for burst cycles to occur. Cycles such as 64-bit floating-point operand reads will burst if BRDY is returned to the processor. S0 and S3 track the state of the A3 CPU address output. When a burst read cycle starts, S1 or S2 is entered. The B0MA0 address output will then change its state when MBRDY# and DATASEL are both low. This function is the burst address for bank 0. The B1MA0 address output changes its state when MBRDY# is low and DATASEL is high. This function is the burst address for bank1. The only difference in the two PLDs is the value of DATASEL used to determine the time of which the burst address changes its state.

The S0 and S3 states are required only to ensure that the burst address outputs are valid during the T2 of any read cycle. Figure 17 shows the timing of a burst read

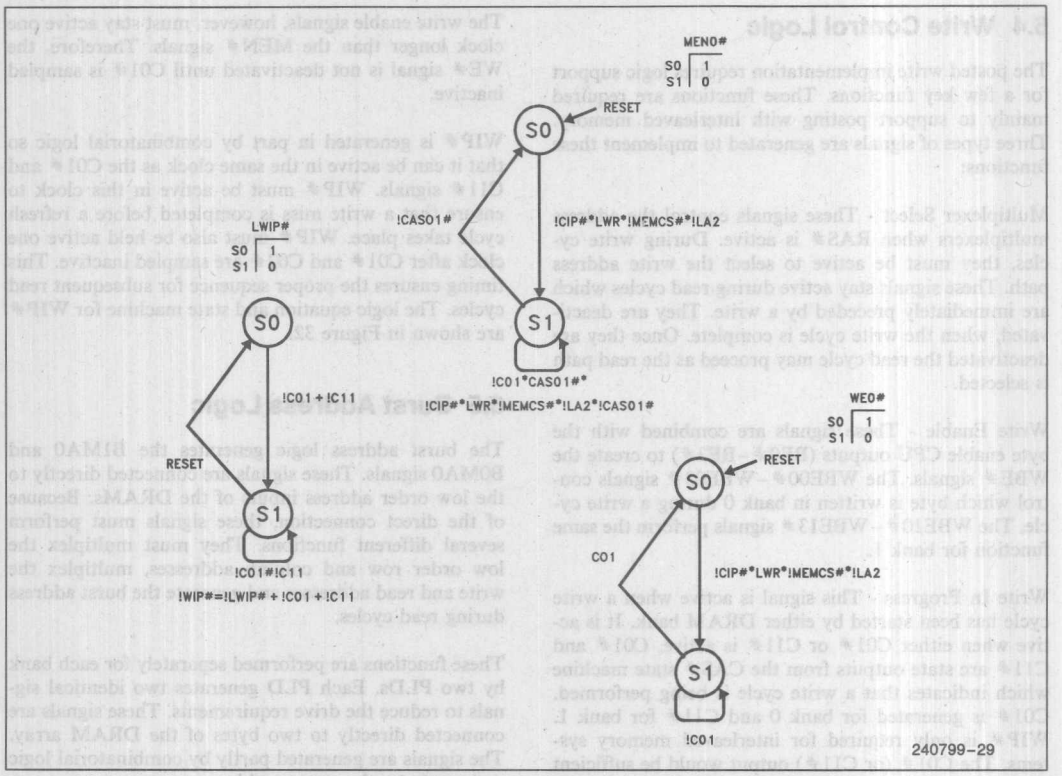


Figure 32. State Machines for MEN0#, WIP#, and WE0#

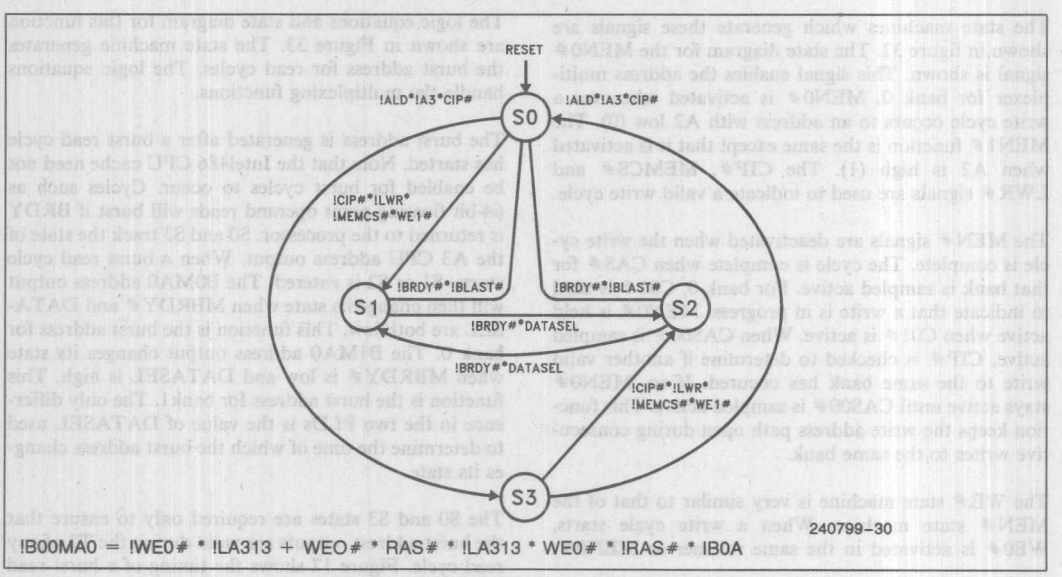


Figure 33. Burst Address Generation

hit cycle. In the first access of this cycle, the burst address must be valid in the first T2 to satisfy the address access time requirements of the DRAM. The value of A3 is sampled with ALD to statisfy this requirement. In this way, the burst address state machine always starts from the correct value of A3. If another wait state is added to this access, this function is not required.

The logic equations which provide the multiplexor function are very simple. The first term of the equations shown in Figure 33 enables the write path. The write enable signals are used to enable this path. When WE0 is active, for example, the value of the multiplexor output is passed through to the DRAM. The second term allows the row address A13 to be passed to the DRAM during a read page miss. This term is also qualified by the write enable signals. In this way, the write address is not disabled early during a read miss. The third term enables the burst address output from the state machine onto the address pins.

7.0 SUMMARY

We have discussed an example memory subsystem for the Intel486 CPU. The material has been presented as a design guide for systems under development or as an optimization for existing systems. We have discussed several key functions which will be summarized in this section. We will also discuss some important timing restrictions. The key functions discussed include an external or second level cache, posted write cycles, and interleaved DRAM banks.

The interleaving technique is used to support the burst bus feature of the Intel486 CPU. The use of this technique allows the DRAM to supply a DWORD every clock during burst cycles. Interleaving proves to be very useful in Intel486 CPU memory designs. Without its use DRAM timings such as tPC (Page Mode Cycle time) and tCP (CAS Precharge time) would prevent zero wait state access at 33 MHz.

Data registers are also used to improve average write cycle latency. These registers hold write data during posted write cycles. Write posting can improve average write latency to under 3 clocks for many applications. This improvement is important in Intel486 CPU based systems because 65% to 70% of all bus cycles are writes. Without using a latency improvement technique such as write posting average write latency will be above 5 clocks.

The write posting technique also improves memory performance in other ways. Write cycles, particularly DRAM page misses, can be overlapped with read hit cycles in the second level cache. This fact greatly reduces the delay caused by read cycles which immediatly follow write cycles.

Analysis of this memory subsystem design has shown that use of these features has resulted in a low latency response to the CPU. Over several important applications the following characteristics have been recorded. The average clock cycles required to complete the first read is 3.5 clocks. Subsequent cycles of a burst are always processed in one clock. Write cycles average 2.5 clocks. These average counts result from the following DRAM access rates. Read accesses from the cache always occur in zero wait states.

Table 3. Dram Function Latencies

DRAM Function	First Access Burst	Subsequent Burst Accesses	Write Cycles
Page Hit	3	1	2
Page Miss	7	1	5*

NOTE:
*Write miss latencies occur only during cycles subsequent to a write miss cycle.

7.1 Timing Restrictions

A few DRAM timing restrictions must be mentioned. These timings become critical at 33 MHz. These timings are critical due primarily to the latency of the first cycle of a read page hit. Since three clocks are used the following timing restrictions exist.

tRAC = Data access time from RAS# active

tCAA = Data access time from column address valid

tCAC = Data access time from CAS# active

tRP = RAS# precharge time

At 33 MHz

tRAC = 71.5 ns

tCAA = 37.5 ns

tCAC = 34 ns

tRP = 60.6 ns

At 25 MHz

tRAC = 101.5 ns

tCAA = 51 ns

tCAC = 61.5 ns

tRP = 80 ns

APPENDIX A
PLD CODES AND SCHEMATICS

A.1 PLD DEVICES

Many design examples in this manual use PLDs (Programmable Logic Devices) which can be programmed by the user to implement random logic. A PLD device can be used as a state machine or a signal decoder, for example. The advantages of PLDs include the following:

1. PLD pinout is determined by the designer, which can simplify board layout by moving signals as required.
2. PLDs are inexpensive as compared to dedicated bus controllers.

Intel EPLDs (Erasable Programmable Logic Devices) have the following additional advantages:

1. Programmability/erasability allows EPLD functions to be changed easily, simplifying prototype development.
2. Since EPLDs are implemented in CMOS technology, they can consume an order of magnitude less power than bipolar PLDs. Power-conscious applications can benefit greatly from using EPLDs.
3. Since the EPROM cell size is an order of magnitude smaller than an equivalent bipolar fuse, EPLDs can implement more functions in the same package. This higher integration can result in a lower overall component count for a design. The added flexibility can also mean that an extremely low number of "raw" (unprogrammed) devices need to be stocked versus bipolar PLDs.
4. Once an EPLD design has been tested, plastic OTP (One-Time Programmable) versions of the device can be used in a production environment.

PLDs have the following tradeoffs:

1. Most PLDs do not have buried (not connected to outputs) registers. For some state machine applications, this means using an otherwise available output pin to store the current state.
2. The drive capability of CMOS EPLDs may be insufficient for some applications. While the trend is towards use of CMOS throughout a system, in cases where high current levels are required, some additional buffering may be required with EPLDs.

A PLD consists logically of a programmable AND array whose output terms feed a fixed OR array. Any sum-of-products equations, within the limits of the number of PLD inputs, outputs, and equation terms, can be realized by specifying the correct AND array connections. Figure A-1 shows an example of two PLD equations and the corresponding logic array. Note that every horizontal line in the AND array represents a multi-input AND gate; every vertical line represents a possible input to the AND gate. An X at the intersection of a horizontal line and a vertical line represents a connection from the input to the AND gate.

The sum-of-products is then routed to a configurable macrocell. The macrocell in Figure A-2 can be configured as a combinational output or registered output. The output can be active high or active low. A separate AND term controls the output buffer.

Designing with PLDs consists of determining where Xs must be placed in the AND array and how to configure the macrocell. This task is simplified by logic compilers, such as iPLS II (Intel's Programmable Logic Software II) or ABEL. Logic compilers accept input in the form of sum-of-product equations and translate the input into a JEDEC programming file that can be used by programming hardware/software.

Intel PLDs are described in the *Programmable Logic Handbook*. Three Intel PLDs have been used in this manual to implement state machine and decode functions. These PLDs include:

- 85C220—fast 20-pin superset of 16 x 8 type bipolar and CMOS PLDs.
- 85C224—fast 24-pin superset of 20 x 8 type bipolar and CMOS PLDs.
- 85C508—fast address decode PLD with integral transparent latches.

The 85C220 and 85C224 PLDs are both available at clock speeds to support fast state-machines in Intel486 systems. The 85C508 provides a fast Enable-to-Output time with a minimal system setup time.

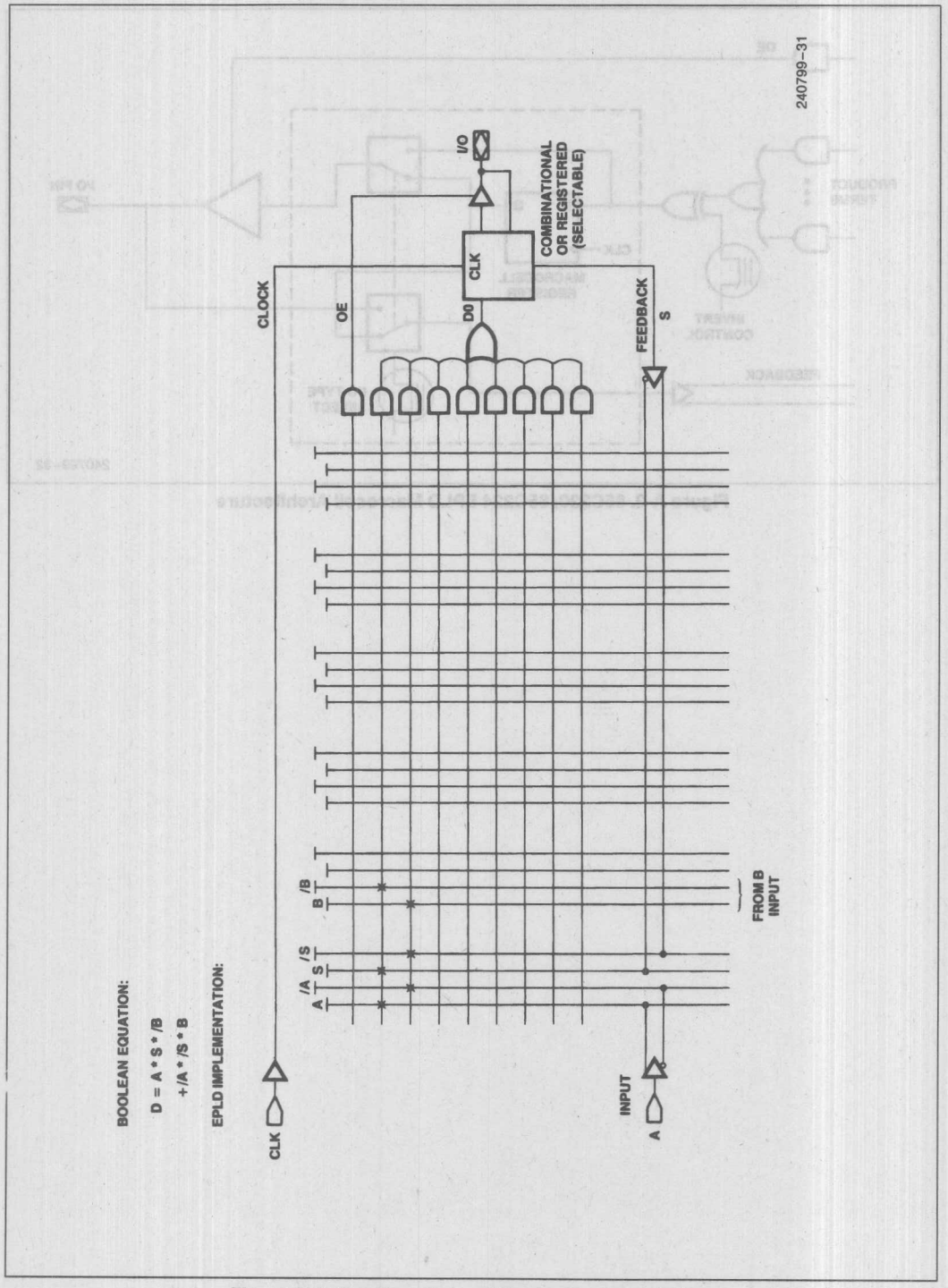


Figure A-1. PLD Equation and Device Implementation

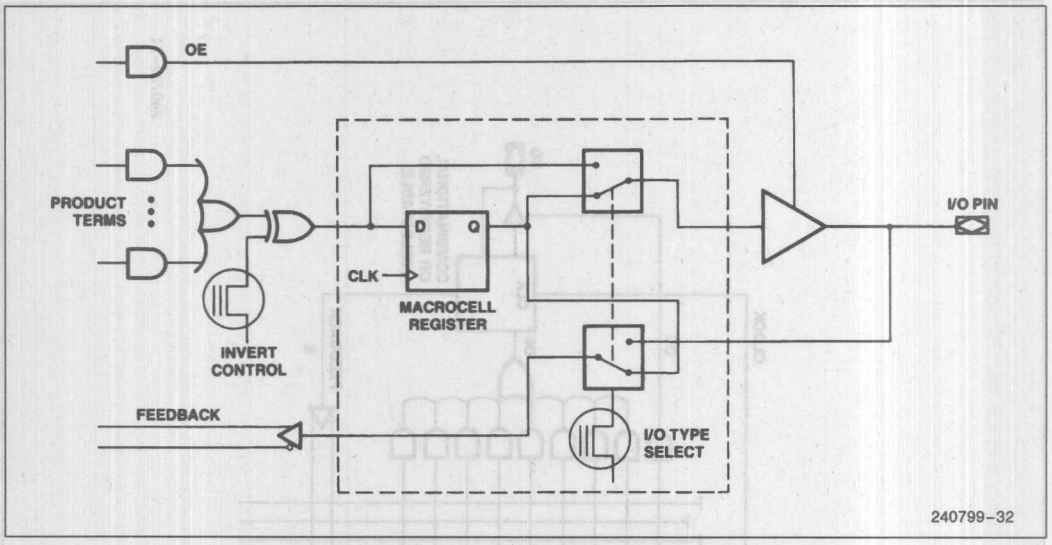


Figure A-2. 85C220/85C224 EPLD Macrocell Architecture

```
module          SC_MODE_DRAM_CTRL_1          flag '-r4'

title   'DRAM CONTROLLER - PLD 1, INTEL CORPORATION'
" Cycle Tracking Logic
" Implemented with Intel 85C224 EPLD

    SC1    device      'E224';

    x       =      .X.;          " ABEL 'don't care' symbol
    c       =      .C.;          " ABEL 'clocking input' symbol

" Inputs

    CLK     pin    1;     "i486 CPU input CLK"
    BLAST~  pin    2;     "i486 CPU BLAST# output
    MEMCS~  pin    3;     "Memory Chip Select
    AHOLD   pin    4;     "Address HOLD input to i486"
    HIT~    pin    5;     "DRAM Page Hit Signal
    BOFF~   pin    6;     "Backoff input to i486"
    ADS~    pin    7;     "Address Status output of i486"
    RFRQ    pin    8;     "Refresh Request Signal
    RESET   pin    9;     "System Reset
    BRDY~   pin   10;     "Processor Burst Ready pin
    MRDY~   pin   11;     "Memory ready
    RAS~    pin   14;     "Row Address Strobe
    EP      pin   23;     "Refresh indicator - count on RAS~ low

" Output

    RFACK~  pin   15;     " Refresh acknowledge
    CIP~    pin   16;     " ADS~ active indicator - Cycle OK
    M~      pin   17;     " Miss Indicator
    CT      pin   18;     " Cycle Track Output
    PCHG    pin   19;     " Precharge state indicator
    Q1      pin   20;     " Precharge state indicator
    ALD     pin   21;     " Address Latch Disable
    adlst~  pin   22;     " State Variable


state_diagram [CIP~, M~]

    state [1, 1]: if RESET then [1, 1] else
                  if AHOLD # !RFACK~ # EP then [1, 1] else
                  if !ADS~ # CT then [0, 1] else [1, 1];

    state [0, 1]: if RESET # !BOFF~ # MEMCS~ then [1, 1] else
                  if HIT~ & !RAS~ & !MRDY~ then [0, 0] else
                  if (!MRDY~ # (!BRDY~ & !BLAST~)) then [1, 1]
                  else [0, 1];

    state [0, 0]: if RESET # !BOFF~ then [1, 1] else
                  if !PCHG & (CT # !ADS~) then [0, 1] else
                  if !PCHG & !CT then [1, 1] else
                  [0, 0];

    state [1, 0]: goto [1, 1];

state_diagram [PCHG, Q1]

    state [0, 0]:  if RESET then [0, 0] else
                  if !RAS~ then [1, 0] else
                  if RAS~ & !RFACK~ then [0, 1] else [0, 0];

    state [1, 0]:  if RESET then [0, 0] else
```

240799-33

```
                    if RAS~ & !EP then [0, 0] else
                    if RFACK~ & EP & !RAS~ then [1, 1] else
                    if RAS~ & EP then [0, 1] else [1, 0];

    state [0, 1]:  goto [1, 0];

    state [1, 1]:  goto [0, 0];

state_diagram [CT]

    state [0]:     if RESET then [0] else
                   if !ADS~ & (AHOLD # !RFACK~ # !M~ # EP)then [1] else [0];

    state [1]:     if RESET # !BOFF~ then [0] else
                   if !CIP~ & M~ then [0] else [1];

state_diagram [RFACK~]

    state[1]:      if RESET then [1] else
                   if !CIP~ & RFRQ & !MRDY~ & !HIT~ then [0] else
                   if !CIP~ & RFRQ & (!BRDY~ & !BLAST~) #
                   RFRQ & CIP~ & ADS~ then [0] else [1];

    state[0]:      if RESET # !BOFF~ then [1] else
                   if RAS~ then [1] else [0];

state_diagram [ALD, adlst~]

     state [0, 1]: if RESET then [0, 1] else
                   if !ADS~ # !CIP~ & !MEMCS~ then [1, 0] else [0, 1];

     state [1, 0]: if RESET then [0, 1] else
                   if !CIP~ & MEMCS~ then [0, 1] else
                   if HIT~ & !MRDY~ then [1, 1] else
                   if !HIT~ & !MRDY~ then [0, 1] else
                   if !BRDY~ & !BLAST~ then [0, 1] else [1, 0];

     state [1, 1]: if RESET  then [0, 1] else
                   if !CIP~ & (!PCHG # MEMCS~) then [0, 1] else [1, 1];

     state [0, 0]: goto [0, 1];

test_vectors

    ([CLK,BLAST~,MEMCS~,AHOLD,HIT~,BOFF~,ADS~,RFRQ,RESET,BRDY~,MRDY~,RAS~,EP] ->
     [RFACK~, CIP~,M~,CT,PCHG,Q1,ALD,adlst~])

"    C  B  M  A  H  B  A  R  R  B  R  R  E       R  C  M  C  P  Q  A  a
"    L  L  E  H  I  O  D  F  E  R  D  A  P       F  I  ~  T  C  1  L  d
"    K  A  M  O  T  F  S  R  S  D  Y  S  ~       A  P             H  D  l
"       S  C  L  ~  F  ~  Q  E  Y  ~  ~          C  ~             G     s
"       T  S  D     ~        T  ~                K                      t
"       ~  ~                                                           ~
"
"
"

    [c, x, x, x, x, x, 1, x, 1, x, x, x, x]  -> [1, 1, 1, 0, 0, 0, 0, 1];
    [c, x, x, x, x, x, 1, x, 1, x, x, x, x]  -> [1, 1, 1, 0, 0, 0, 0, 1];
    [c, 1, x, 0, x, 1, 1, 0, 0, 1, 1, 1, 0]  -> [1, 1, 1, 0, 0, 0, 0, 1];
    [c, 1, x, 0, x, 1, 0, 0, 0, 1, 1, 1, 0]  -> [1, 0, 1, 0, 0, 0, 1, 0];
    [c, 1, x, 0, x, 1, 1, 0, 0, 1, 1, 1, 0]  -> [1, 0, 1, 0, 0, 0, 1, 0];
    [c, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0]  -> [1, 0, 1, 0, 0, 0, 1, 0];
    [c, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0]  -> [1, 0, 1, 0, 1, 0, 1, 0];
                                                                          240799-34
    [c, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0]  -> [1, 0, 1, 0, 1, 0, 1, 0];
    [c, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0]  -> [1, 0, 1, 0, 1, 0, 1, 0];
    [c, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0]  -> [1, 0, 1, 0, 1, 0, 1, 0];
    [c, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0]  -> [1, 0, 1, 0, 1, 0, 1, 0];
    [c, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0]  -> [1, 1, 1, 0, 1, 0, 0, 1];
    [c, x, x, 0, x, 1, 1, 0, 0, 1, 1, 0, 0]  -> [1, 1, 1, 0, 1, 0, 0, 1];

end SC_MODE_DRAM_CTRL_1;
```

240799-35

```
module          PG_MODE_DRAM_CTRL_2          flag '-r4'

title   'DRAM CONTROLLER - PLD 2, INTEL CORPORATION'
" This PLD generates CAS0
" Implemented with the Intel 85C220 EPLD

    SC2     device      'E0320';

    x       =     .X.;          " ABEL 'don't care' symbol
    c       =     .C.;          " ABEL 'clocking input' symbol

" Inputs

    CLK     pin   1;    "i486 CPU input CLK"
    RFACK   pin   2;    "Refresh Acknowledge
    CIP~    pin   3;    "Cycle OK
    LA2     pin   4;    "Latched A2
    HIT~    pin   5;    "DRAM Page Hit Signal
    BOFF~   pin   6;    "Backoff input to i486"
    LW_R~   pin   7;    "Latched W/R#
    RAS~    pin   8;    "Row Address Strobe
    RESET   pin   9;    "System Reset
    RDY~    pin  12;    "Processor RDY#
    MEMCS~  pin  13;    "Memory Chip Select
    BRDY~   pin  18;    "Processor BRDY#
    BLAST~  pin  19;    "Processor BLAST#

" Output

    CAS00~  pin  14;    "CAS0 byte 0,2
    C1      pin  15;    "state variable
    C2      pin  16;    "state variable
    CAS01~  pin  17;    "CAS0 byte 1,3

state_diagram [CAS10~,CAS11~, C1, C2]

    state [1, 1, 1, 1]: if RESET # !BOFF~ then [1, 1, 1, 1] else
                        if !RFACK & !CIP~ & !LA2 & LW_R~ & !MEMCS~ then
                        [1, 1, 0, 1] else if !RFACK & !CIP~ & !LW_R~ & !RAS~
                        & !HIT~ & !MEMCS~ # (RFACK & RAS~) then
                        [0, 0, 1, 1] else [1, 1, 1, 1];

    state [1, 1, 0, 1]: if RESET # !BOFF~ then [1, 1, 1, 1] else
                        if !RAS~ & RDY~ then [0, 0, 0, 0] else [1, 1, 0, 1];

    state [0, 0, 0, 0]: if RESET # !BOFF~ then [1, 1, 1, 1] else
                        if !CIP~ & !LA2 & LW_R~ & !MEMCS~ then [1, 1, 0, 0] else
                        if CIP~ # (!CIP~ & (MEMCS~ # !LW_R~)) # (!CIP~ & LW_R~
                        & LA2) then [1, 1, 1, 1] else [0, 0, 0, 0];

    state [1, 1, 0, 0]: if !RAS~ then  [0, 0, 0, 0] else [1, 1, 0, 0];

    state [0, 0, 1, 1]: if RESET # !BOFF~ then [1, 1, 1, 1] else
                        if !BRDY~ & !BLAST~ & !RFACK then [1, 1, 1, 1] else
                        if !BRDY~ & BLAST~ & !LA2 then [1, 1, 1, 0] else
                        if !BRDY~ & BLAST~ & LA2 then [0, 0, 1, 0] else
                        if RFACK then [0, 0, 1, 0] else
                        if BRDY~ & !RFACK then [0, 0, 1, 1];

    state [1, 1, 1, 0]: if RESET then [1, 1, 1, 1] else
                        if !BOFF~ then [1, 1, 1, 0] else
                        if !BRDY~ & BLAST~ then [0, 0, 1, 1] else
                        if !BRDY~ & !BLAST~ then [1, 1, 1, 1] else
                        [1, 1, 1, 0];
```

240799-36

```
    state [0, 0, 1, 0]: if RESET # !BOFF~ then [1, 1, 1, 1] else
                        if !BRDY~ & BLAST~ then [1, 1, 1, 0] else
                        if !BRDY~ & !BLAST~ # BRDY~ then [1, 1, 1, 1] ;

test_vectors

    ([CLK,RFACK,CIP~,LA2,HIT~,BOFF~,LW_R~,RAS~,RESET,RDY~,MEMCS~,BRDY~,BLAST~]
    ->          [CAS00~,C1,C2,CAS01~])

"    C  R  C  L  H  B  L  R  R  R  M  B  B       C  C  C  C
"    L  F  I  A  I  O  W  A  E  D  E  R  L       A  1  2  A
"    K  A  P  2  T  F  R  S  S  Y  M  D  A       S        S
"       C  ~     ~  F  ~  ~  E  ~  C  Y  S       0        0
"       K        ~        T     S     T       0        1
"                                  ~     ~       ~        ~
"
"
"

    [c, x, x, x, x, x, 1, x, 1, x, x, x, x]  -> [x, x, x, x];
    [c, x, 1, 0, x, x, 1, x, 1, x, x, x, x]  -> [1, 1, 1, 1];
    [c, 0, 1, 0, x, 1, 1, 0, 0, 1, 0, 1, 1]  -> [1, 1, 1, 1];
    [c, 0, 1, 0, x, 1, 1, 0, 0, 1, 0, 1, 1]  -> [1, 1, 1, 1];
    [c, 0, 1, 0, x, 1, 1, 0, 0, 1, 0, 1, 1]  -> [1, 1, 1, 1];
    [c, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1]  -> [1, 0, 1, 1];
    [c, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1]  -> [1, 0, 1, 1];
    [c, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1]  -> [0, 0, 0, 0];
    [c, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1]  -> [1, 0, 0, 1];
    [c, 0, 1, x, 0, 1, 1, 0, 0, 1, 0, 1, 1]  -> [0, 0, 0, 0];
    [c, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1]  -> [1, 0, 0, 1];
    [c, 0, 1, x, 0, 1, 1, 0, 0, 1, 0, 1, 1]  -> [0, 0, 0, 0];
    [c, 0, 1, 1, x, 1, 1, 0, 0, 1, 0, 1, 1]  -> [1, 1, 1, 1];

end PG_MODE_DRAM_CTRL_2;
```

240799-37

```
module          SC_MODE_DRAM_CTRL_3          flag '-r4'

title   'DRAM CONTROLLER - PLD 3, INTEL CORPORATION'
" This PLD generates RAS
" Implemented with the Intel 85C220 EPLD

    SC3    device       'E0320';

    x       =     .X.;          " ABEL 'don't care' symbol
    c       =     .C.;          " ABEL 'clocking input' symbol

" Inputs

    CLK     pin   1;     "i486 CPU input CLK"
    M~      pin   2;     "Write Miss Indicator
    CIP~    pin   3;     "Cycle OK
    MEMCS~  pin   4;     "Memory Chip Select
    HIT~    pin   5;     "DRAM Page Hit Signal
    RFACK   pin   6;     "Refresh Acknowledge
    PCHG    pin   7;     "RAS precharge count
    WIP~    pin   8;     "Write in Progress
    RESET   pin   9;     "System Reset
    Q1      pin  12;     "RAS refresh count

" Output

    RAS2~   pin  13;     "
    RAS1~   pin  14;     " RAS byte 0,2
    EP      pin  15;     " state variable
    EP1     pin  16;     " state variable
    RAS0~   pin  17;     " RAS byte 1,3
    RAS3~   pin  18;     "
    CSWIP~  pin  19;     "

state_diagram [RAS0~,RAS1~,EP]

    state [1, 1, 0]:     if RESET then [1, 1, 0] else
                         if !CIP~ & !CSWIP~ & !PCHG then [0, 0, 0] else
                         if RFACK & WIP~ then [1, 1, 1] else
                         [1, 1, 0];

    state [0, 0, 0]:     if RESET then [1, 1, 0] else
                         if RFACK  then [0, 0, 1] else
                         if !CIP~ & HIT~ & !MEMCS~ then [1, 1, 0]
                         else [0, 0, 0];

    state [0, 0, 1]:     if RESET then [1, 1, 0] else
                         if !RFACK & !PCHG  then [1, 1, 0] else
                         if RFACK & !WIP~ # !RFACK & PCHG then
                         [0, 0, 1] else if RFACK & WIP~ & !Q1 then [1, 1, 1];


    state [1, 1, 1]:     if RESET then [1, 1, 0] else
                         if !PCHG then [0, 0, 1] else [1, 1, 1];

    state [0, 1, 0]:     goto [1,1,0];
    state [0, 1, 1]:     goto [1,1,0];
    state [1, 0, 0]:     goto [1,1,0];
    state [1, 0, 1]:     goto [1,1,0];

state_diagram [RAS2~,RAS3~,EP1]

    state [1, 1, 0]:     if RESET then [1, 1, 0] else
                         if !CIP~ & !CSWIP~ & !PCHG then [0, 0, 0] else
```

240799-38

```
                        if RFACK & WIP~ then [1, 1, 1] else [1, 1, 0];

    state [0, 0, 0]:    if RESET then [1, 1, 0] else
                        if RFACK  then [0, 0, 1] else
                        if !CIP~ & HIT~ & !MEMCS~ then [1, 1, 0]
                        else [0, 0, 0];

    state [0, 0, 1]:    if RESET then [1, 1, 0] else
                        if !RFACK & !PCHG  then [1, 1, 0] else
                        if RFACK & !WIP~ # !RFACK & PCHG then
                        [0, 0, 1] else if RFACK & WIP~ & !Q1 then [1, 1, 1];


    state [1, 1, 1]:    if RESET then [1, 1, 0] else
                        if !PCHG then [0, 0, 1] else [1, 1, 1];

    state [0, 1, 0]:    goto [1,1,0];
    state [0, 1, 1]:    goto [1,1,0];
    state [1, 0, 0]:    goto [1,1,0];
    state [1, 0, 1]:    goto [1,1,0];

equations

    !CSWIP~ = (!MEMCS~ # !WIP~)& !RESET;

test_vectors

    ([CLK,M~,CIP~,MEMCS~,HIT~,RFACK,PCHG,WIP~,Q1,RESET] ->
    [RAS0~,RAS1~,EP,RAS2~,RAS3~,EP1])

"    C  M  C  M  H  R  P  W  Q  R        R  R  E  R  R  E
"    L  ~  I  E  I  F  C  I  1  E        A  A  P  A  A  P
"    K     P  M  T  A  H  P     S        S  S     S  S  1
"          ~  C  ~  C  G  ~     E        0  1     2  3
"             S     K           T
"             ~
"
"
"

    [c, x, x, x, x, x, 1, x, x, 1]  -> [x, x, x, x, x, x];
    [c, x, x, x, x, x, 1, x, x, 1]  -> [1, 1, 0, 1, 1, 0];
    [c, x, x, x, x, x, 1, x, x, 1]  -> [1, 1, 0, 1, 1, 0];
    [c, x, x, x, x, x, 1, x, x, 1]  -> [1, 1, 0, 1, 1, 0];
    [c, x, x, x, x, x, 1, x, x, 1]  -> [1, 1, 0, 1, 1, 0];
    [c, x, x, x, x, x, 1, x, x, 1]  -> [1, 1, 0, 1, 1, 0];
    [c, x, x, x, x, x, 1, x, x, 1]  -> [1, 1, 0, 1, 1, 0];
    [c, 1, 1, x, x, 0, 0, 0, 0, 0]  -> [1, 1, 0, 1, 1, 0];
    [c, 1, 1, x, x, 0, 0, 0, 0, 0]  -> [1, 1, 0, 1, 1, 0];
    [c, 1, 0, 0, 0, 0, 0, 0, 0, 0]  -> [0, 0, 0, 0, 0, 0];
    [c, 1, 0, 0, 0, 0, 0, 0, 0, 0]  -> [0, 0, 0, 0, 0, 0];
    [c, 1, 1, x, x, 0, 0, 0, 0, 0]  -> [0, 0, 0, 0, 0, 0];
    [c, 1, 0, 0, 0, 0, 0, 0, 0, 0]  -> [0, 0, 0, 0, 0, 0];
    [c, 1, 1, x, x, 0, 0, 0, 0, 0]  -> [0, 0, 0, 0, 0, 0];
    [c, 1, 0, 0, 0, 0, 0, 0, 0, 0]  -> [0, 0, 0, 0, 0, 0];
    [c, 1, 1, x, x, 0, 0, 0, 0, 0]  -> [0, 0, 0, 0, 0, 0];
    [c, 1, 1, x, x, 0, 0, 0, 0, 0]  -> [0, 0, 0, 0, 0, 0];
    [c, 1, 1, x, x, 0, 0, 0, 0, 0]  -> [0, 0, 0, 0, 0, 0];

end SC_MODE_DRAM_CTRL_3;
```

240799-39

```
module          SC_MODE_DRAM_CTRL_4        flag '-r4'

title   'DRAM CONTROLLER - PLD 4, INTEL CORPORATION'
" This PLD generates MRDY and MBRDY
" Implemented with Intel 85C224 EPLD

    SC4     device      'E224';

    x       =     .X.;          " ABEL 'don't care' symbol
    c       =     .C.;          " ABEL 'clocking input' symbol

" Inputs

    CLK     pin   1;    "i486 CPU input CLK"
    M~      pin   2;    "Miss Indicator
    CIP~    pin   3;    "Cycle OK
    MEMCS~  pin   4;    "Memory Chip Select
    HIT~    pin   5;    "DRAM Page Hit Signal
    RFACK   pin   6;    "Refresh acknowledge"
    ADS~    pin   7;    "CPU ADS~
    W_R     pin   8;    "CPU W/R
    RESET   pin   9;    "System Reset
    dum1    pin  10;    "
    BOFF~   pin  11;    "CPU Backoff input
    WIP~    pin  14;    "Write in Progress
    CAS~    pin  15;    "Column Address Strobe
    BLAST~  pin  22;    "CPU Burst Last output
    RAS~    pin  23;    "Row Address Strobe

" Output

    dum0    pin  16;
    MT      pin  17;    "BRDY state miss tracking
    MRDY~   pin  18;    "Memory RDY (modified with other RDYs)
    DALE~   pin  19;    "Decode Latch enable
    LWR     pin  20;    "Internally latched W/R# for RDY
    BRDY~   pin  21;    "Processor BRDY~

state_diagram [MRDY~]

    state [1]:    if (!RFACK & !ADS~ & W_R & !RAS~ & M~) # (!CIP~ & LWR &
                  !MEMCS~ & !RFACK & M~) then [0] else [1];


    state [0]:    goto [1];

state_diagram [BRDY~, MT]

    state [1, 1]: if !CIP~ & !HIT~ & !MEMCS~ & !LWR & !RFACK & WIP~ & !RAS~
                  then [0, 1] else if !CIP~ & !MEMCS~ & HIT~ & !LWR #
                  !CIP~ & !MEMCS~ & RAS~ & !LWR then [1, 0];

    state [1, 0]: if RESET then [1, 1] else
                  if WIP~ & !RFACK & !CAS~ then [0, 1];

    state [0, 1]: if RESET # !BOFF~ # !BLAST~ then [1, 1] else [0, 1];

state_diagram [DALE~]

    state [0]:    if RESET then [0] else
                  if !ADS~ then [1] else [0];

    state [1]:    if RESET # !BOFF~ then [0] else
                  if !CIP~ then [0] else [1];
```

240799-40

```
state_diagram [LWR]

    state [0]:    if RESET then [0] else
                  if !ADS~ & W_R then [1] else [0];

    state [1]:    if RESET # !BOFF~ then [0] else
                  if !ADS~ & !W_R then [0] else [1];

test_vectors


([CLK,M~,CIP~,MEMCS~,HIT~,RFACK,ADS~,W_R,RESET,WIP~,BOFF~,BLAST~,RAS~]
 ->    [MRDY~,DALE~,LWR,BRDY~])

"    C  M  C  M  H  R  A  W  R  W  B  B  R        M  D  L  B
"    L  ~  I  E  I  F  D  _  E  I  O  L  A        R  A  W  R
"    K     P  M  T  A  S  R  S  P  F  A  S        D  L  R  D
"          ~  C  ~  F  ~     E     F  S  ~        Y  E     Y
"             S     K        T     ~  T           ~  ~     ~
"             ~                    ~
"
"

    [c, x, x, x, x, x, 1, x, 1, x, x, x, x]  -> [x, x, x, x];
    [c, x, 1, 1, x, x, 1, x, 1, x, x, x, x]  -> [1, 0, 0, 1];
    [c, 1, 1, 1, x, 0, 1, x, 0, 0, 1, 1, 1]  -> [1, 0, 0, 1];
    [c, 1, 1, 1, x, 0, 1, x, 0, 0, 1, 1, 1]  -> [1, 0, 0, 1];
    [c, 1, 1, 1, x, 0, 0, 1, 0, 0, 1, 1, 1]  -> [1, 1, 1, 1];
    [c, 1, 0, 0, 0, 0, 1, x, 0, 0, 1, 1, 1]  -> [0, 0, x, 1];
    [c, 1, 0, 0, 0, 0, 1, x, 0, 1, 1, 1, 0]  -> [1, 0, 1, 1];
    [c, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0]  -> [0, 1, 1, 1];
    [c, 1, 0, 0, 0, 0, 1, x, 0, 1, 1, 1, 0]  -> [1, 0, x, 1];
    [c, 1, 1, x, 0, 0, 0, 1, 0, 1, 1, 1, 0]  -> [0, 1, 1, 1];
    [c, 1, 0, 0, 0, 0, 1, x, 0, 1, 1, 1, 0]  -> [1, 0, x, 1];
    [c, 1, 1, x, 0, 0, 1, x, 0, 1, 1, 1, 0]  -> [1, 0, x, 1];
    [c, 1, 1, 1, x, 0, 1, x, 0, 1, 1, 1, 0]  -> [1, 0, x, 1];

end SC_MODE_DRAM_CTRL_4;
```

240799-41

```
module          SC_MODE_DRAM_CTRL_6        flag '-r4'

title   'DRAM CONTROLLER - PLD 6, INTEL CORPORATION'
" This PLD generates A0 for Bank 0
" Implemented with the Intel 85C224 EPLD

    SC6    device      'E224';

    x       =      .X.;          " ABEL 'don't care' symbol
    c       =      .C.;          " ABEL 'clocking input' symbol

" Inputs

    CLK             pin   1;     "i486 CPU input CLK"
    BRDY~           pin   2;     "Burst Ready
    CIP~            pin   3;     "Cycle OK
    MEMCS~          pin   4;     "Memory Chip Select
    LA313           pin   5;     "Latched A3 and A13
    DATASEL         pin   6;     "Bank Select for Reads
    RAS~            pin   7;     "Row address strobe
    LW_R            pin   8;     "CPU W/R latched~
    RESET           pin   9;     "System Reset
    BLAST~          pin  10;     "CPU BLAST~ output
    A3              pin  11;     "CPU A3 line
    ALD             pin  14;     "Address Latch disable
    dum1            pin  15;
    WE0~            pin  22;     "Write enable
    dum2            pin  23;

" Output

    B00MA0          pin  21;     "Bank 0 A0
    B0A             pin  20;
    CS0~            pin  19;
    dun             pin  18;
    dum             pin  17;
    B01MA0          pin  16;     "Bank 0 A0

state_diagram [B0A, CS0~]

    state [1, 1]: if RESET then [1, 1] else
                  if CIP~ & !ALD & !A3 then [0, 1] else
                  if !CIP~ & !ALD & !A3 then [0, 1] else
                  if !CIP~ & !LW_R & !MEMCS~ & WE0~ then [1, 0] else [1, 1];

    state [0, 1]: if RESET then [1, 1] else
                  if CIP~ & !ALD & A3 then [1, 1] else
                  if !CIP~ & !ALD & A3 then [1, 1] else
                  if !CIP~ & !LW_R & !MEMCS~ & WE0~ then [0, 0] else [0, 1];


    state [1, 0]: if RESET #  (!BRDY~ & !BLAST~) then [1, 1] else
                  if !BRDY~ & !DATASEL then [0, 0] else [1, 0];

    state [0, 0]: if RESET #  (!BRDY~ & !BLAST~) then [1, 1] else
                  if !BRDY~ & !DATASEL then [1, 0] else [0, 0];

equations

!B00MA0 = !WE0~ & !LA313 # WE0~ & RAS~ & !LA313 # WE0~ & !RAS~ & !B0A;

!B01MA0 = !WE0~ & !LA313 # WE0~ & RAS~ & !LA313 # WE0~ & !RAS~ & !B0A;

end SC_MODE_DRAM_CTRL_6;
```

240799-42

```
module          SC_MODE_DRAM_CTRL_7      flag '-r4'

title   'DRAM CONTROLLER - PLD 7, INTEL CORPORATION'
" This PLD generates DATASEL and WE
" Implemented with Intel 85C220 EPLD

    SC7    device      'E0320';

    x       =     .X.;          " ABEL 'don't care' symbol
    c       =     .C.;          " ABEL 'clocking input' symbol

" Inputs

    CLK             pin   1;    "i486 CPU input CLK"
    BRDY~           pin   2;    "Burst Ready
    CIP~            pin   3;    "Cycle OK
    MEMCS~          pin   4;    "Memory Chip Select
    LA2             pin   5;    "Latched A2
    CAS00~          pin   6;    "CAS output Bank0
    CAS10~          pin   7;    "CAS output Bank1
    LW_R            pin   8;    "Latched CPU W/R
    RESET           pin   9;    "System Reset
    BLAST~          pin  12;    "CPU Burst Last output
    BOFF~           pin  13;    "CPU Backoff input
    HIT~            pin  19;    "DRAM page hit signal

" Output

    DATASEL         pin  14;    "Bank select for reads
    RS~             pin  15;    "State variable
    RALE~           pin  16;    "Enable Row Address Latch
    WE~             pin  17;    "Write Enable posted writes
    BSEL            pin  18;    "Selects read or write data path


state_diagram [DATASEL, RS~]

    state [1, 1]:       if RESET then [1, 1] else
                        if !CIP~ & !LA2 & !LW_R & !MEMCS~ then [0, 0] else
                        if !CIP~ & LA2 & !LW_R & !MEMCS~ then [1, 0] else [1, 1];

    state [1, 0]:       if RESET # !BOFF~ # (!BRDY~ & !BLAST~) then [1, 1] else
                        if !BRDY~ & BLAST~ then [0, 0] else [1, 0];

    state [0, 0]:       if RESET # !BOFF~ # (!BRDY~ & !BLAST~) then [1, 1] else
                        if !BRDY~ & BLAST~ then [1, 0] else [0, 0];

    state [0, 1]:       goto [1,1];


state_diagram [WE~]

    state [1]:          if RESET then [1] else
                        if LW_R & !CIP~ & !MEMCS~ then [0] else [1];

    state [0]:          if RESET # !BOFF~ then [1] else
                        if LW_R & !CIP~ & !MEMCS~ then [0] else
                        if CAS00~ + CAS10~ then [1];

state_diagram [RALE~]

    state [0]:          if RESET then [0] else
                        if !CIP~ & HIT~ & !MEMCS~ then [1] else [0];
```

240799-43

```
    state [1]:          if RESET # !BOFF~ then [0] else
                        if !HIT~ then [0] else [1];

end SC_MODE_DRAM_CTRL_87;
```

240799-44

```
module          SC_MODE_DRAM_CTRL_8          flag '-r4'

title   'DRAM CONTROLLER - PLD 8, INTEL CORPORATION'
" This PLD generates CAS1 (CAS for Bank 1)

    SC8    device       'E0320';

    x       =      .X.;          " ABEL 'don't care' symbol
    c       =      .C.;          " ABEL 'clocking input' symbol

" Inputs

    CLK     pin    1;     "i486 CPU input CLK
    RFACK   pin    2;     "Refresh Acknowledge
    CIP~    pin    3;     "Cycle OK
    LA2     pin    4;     "Latched A2.
    HIT~    pin    5;     "DRAM Page Hit Signal
    BOFF~   pin    6;     "Backoff input to i486"
    LW_R~   pin    7;     "Latched W/R#
    RAS~    pin    8;     "Row Address Strobe
    RESET   pin    9;     "System Reset
    RDY~    pin   12;     "Processor RDY#
    MEMCS~  pin   13;     "Memory Chip Select
    BRDY~   pin   18;     "Processor BRDY#
    BLAST~  pin   19;     "Processor BLAST#

" Output

    CAS10~  pin   14;     "CAS1 byte 0,2
    C1      pin   15;     "state variable
    C2      pin   16;     "state variable
    CAS11~  pin   17;     "CAS1 byte 1,3

state_diagram [CAS10~,CAS11~, C1, C2]

    state [1, 1, 1, 1]: if RESET # !BOFF~ then [1, 1, 1, 1] else
                        if !RFACK & !CIP~ & LA2 & LW_R~ & !MEMCS~ then
                        [1, 1, 0, 1] else if !RFACK & !CIP~ & !LW_R~ & !RAS~
                        & !HIT~ & !MEMCS~ # (RFACK & RAS~) then
                        [0, 0, 1, 1] else [1, 1, 1, 1];

    state [1, 1, 0, 1]: if RESET # !BOFF~ then [1, 1, 1, 1] else
                        if !RAS~ & RDY~ then [0, 0, 0, 0] else
                        [1, 1, 0, 1];

    state [0, 0, 0, 0]: if RESET # !BOFF~ then [1, 1, 1, 1] else
                        if !CIP~ & LA2 & LW_R~ & !MEMCS~ then [1, 1, 0, 0] else
                        if CIP~ # (!CIP~ & (MEMCS~ # !LW_R~)) # (!CIP~ & LW_R~
                        & !LA2) then [1, 1, 1, 1] else [0, 0, 0, 0];

    state [1, 1, 0, 0]: if !RAS~ then  [0, 0, 0, 0] else [1, 1, 0, 0];

    state [0, 0, 1, 1]: if RESET # !BOFF~ then [1, 1, 1, 1] else
                        if !BRDY~ & !BLAST~ & !RFACK then [1, 1, 1, 1] else
                        if !BRDY~ & BLAST~ & LA2 then [1, 1, 1, 0] else
                        if !BRDY~ & BLAST~ & !LA2 then [0, 0, 1, 0] else
                        if RFACK then [0, 0, 1, 0] else
                        if BRDY~ & !RFACK then [0, 0, 1, 1];

    state [1, 1, 1, 0]: if RESET then [1, 1, 1, 1] else
                        if !BOFF~ then [1, 1, 1, 0] else
                        if !BRDY~ & BLAST~ then [0, 0, 1, 1] else
                        if !BRDY~ & !BLAST~ then [1, 1, 1, 1] else
                        [1, 1, 1, 0];
```

240799-45

```
    state [0, 0, 1, 0]: if RESET # !BOFF~ then [1, 1, 1, 1] else
                        if !BRDY~ & BLAST~ then [1, 1, 1, 0] else
                        if !BRDY~ & !BLAST~ # BRDY~ then [1, 1, 1, 1] ;


test_vectors

    ([CLK,RFACK,CIP~,LA2,HIT~,BOFF~,LW_R~,RAS~,RESET,RDY~,MEMCS~,BRDY~,BLAST~]
    ->          [CAS10~,C1,C2,CAS11~])

"    C  R  C  L  H  B  L  R  R  R  M  B  B       C  C  C  C
"    L  F  I  A  I  O  W  A  E  D  E  R  L       A  1  2  A
"    K  A  P  2  T  F  R  S  S  Y  M  D  A       S           S
"       C  ~     ~  F  ~  ~  E  ~  C  Y  S       0           0
"       K           ~        T     S     T       0           1
"                                  ~     ~       ~           ~
"
"
"

    [c, x, x, x, x, x, 1, x, 1, x, x, x, x]  -> [x, x, x, x];
    [c, x, 1, 1, x, x, 1, x, 1, x, x, x, x]  -> [1, 1, 1, 1];
    [c, 0, 1, 1, x, 1, 1, 0, 0, 1, 0, 1, 1]  -> [1, 1, 1, 1];
    [c, 0, 1, 1, x, 1, 1, 0, 0, 1, 0, 1, 1]  -> [1, 1, 1, 1];
    [c, 0, 1, 1, x, 1, 1, 0, 0, 1, 0, 1, 1]  -> [1, 1, 1, 1];
    [c, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1]  -> [1, 0, 1, 1];
    [c, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1]  -> [1, 0, 1, 1];
    [c, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1]  -> [0, 0, 0, 0];
    [c, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1]  -> [1, 0, 0, 1];
    [c, 0, 1, x, 0, 1, 1, 0, 0, 1, 0, 1, 1]  -> [0, 0, 0, 0];
    [c, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1]  -> [1, 0, 0, 1];
    [c, 0, 1, x, 0, 1, 1, 0, 0, 1, 0, 1, 1]  -> [0, 0, 0, 0];
    [c, 0, 1, 0, x, 1, 1, 0, 0, 1, 0, 1, 1]  -> [1, 1, 1, 1];

end SC_MODE_DRAM_CTRL_8;
```

240799-46

```
module          SC_MODE_DRAM_CTRL_11        flag '-r4'

title   'DRAM CONTROLLER - PLD 11, INTEL CORPORATION'
" This PLD generates the mux enables, write enables and WIP#
" Implemented with the Intel 85C220 EPLD

      SC11   device      'E0320';

    x       =     .X.;          " ABEL 'don't care' symbol
    c       =     .C.;          " ABEL 'clocking input' symbol

" Inputs

    CLK     pin   1;    "i486 CPU input CLK"
    LA2     pin   2;    "Latched A2.
    CIP~    pin   3;    "Cycle OK
    MEMCS~  pin   4;    "Memory Chip select.
    RESET   pin   5;    "System Reset
    LW_R    pin   6;    "latched CPU W/R#
    C01     pin   7;    "Write indication Bank0
    CAS01~  pin   8;    "CAS output Bank 0
    C11     pin   9;    "Write indication Bank1
    CAS11~  pin  19;    "CAS output Bank 1

" Output

    WIP~    pin  12;    "Write in Progress
    MEN0~   pin  13;    "Mux enable Bank 0
    WE0~    pin  14;    "Write enable Bank 0
    LWIP~   pin  15;    "Latched WIP~
    dum     pin  16;    "
    WE1~    pin  17;    "Write enable Bank 1
    MEN1~   pin  18;    "Mux enable Bank1

state_diagram [WE0~]

    state [1]:    if RESET then [1] else
                  if !CIP~ & LW_R & !MEMCS~ & !LA2 then [0];

    state [0]:    if RESET then [1] else
                  if !C01 then [0] else
                  if C01 then [1];

state_diagram [WE1~]

    state [1]:    if RESET then [1] else
                  if !CIP~ & LW_R & !MEMCS~ & LA2 then [0];

    state [0]:    if RESET then [1] else
                  if !C11 then [0] else
                  if C11 then [1];

state_diagram [LWIP~]

    state [1]:    if !C01 # !C11 then [0] else [1];


    state [0]:    if RESET then [1] else
                  if !C01 # !C11 then [0] else [1];

state_diagram [MEN0~]

    state [1]:    if RESET then [1] else
                  if !CIP~ & LW_R & !MEMCS~ & !LA2 then [0];
```

240799-47

```
    state [0]:    if RESET then [1] else
                  if !C01 & CAS01~ then [0] else
                  if !CIP~ & LW_R & !MEMCS~ & !LA2 & !CAS01~ then [0] else
                  if !CAS01~ then [1];

state_diagram [MEN1~]

    state [1]:    if RESET then [1] else
                  if !CIP~ & LW_R & !MEMCS~ & LA2 then [0];

    state [0]:    if RESET then [1] else
                  if !C11 & CAS11~ then [0] else
                  if !CIP~ & LW_R & !MEMCS~ & !LA2 & !CAS11~ then [0] else
                  if !CAS11~ then [1];

equations

     !WIP~ = !LWIP~ # !C01 # !C11;

end SC_MODE_DRAM_CTRL_11;
```

240799-48

```
module          SC_MODE_DRAM_CTRL_12        flag '-r4'

title   'DRAM CONTROLLER - PLD 12, INTEL CORPORATION'
" This PLD generates the WRE write enables
" Implemented with the Intel 85C220 EPLD

    SC12   device       'E0320';

    x       =      .X.;          " ABEL 'don't care' symbol
    c       =      .C.;          " ABEL 'clocking input' symbol

" Inputs

    nc0             pin   1;     "no connect
    WE0~            pin   2;     "Write Enable Bank 0
    WE1~            pin   3;     "Write Enable Bank 1
    BE0~            pin   4;     "Byte Enable 0
    BE1~            pin   5;     "Byte Enable 1
    BE2~            pin   6;     "Byte Enable 2
    BE3~            pin   7;     "Byte Enable 3
    nc1             pin   8;     "
    nc2             pin   9;     "

" Output

    WRE00~          pin  12;
    WRE10~          pin  13;
    WRE01~          pin  14;
    WRE11~          pin  15;
    WRE02~          pin  16;
    WRE12~          pin  17;
    WRE03~          pin  18;
    WRE13~          pin  19;

equations

   !WRE00~ = !WE0~ & !BE0~;

   !WRE10~ = !WE1~ & !BE0~;

   !WRE01~ = !WE0~ & !BE1~;

   !WRE11~ = !WE1~ & !BE1~;

   !WRE02~ = !WE0~ & !BE2~;

   !WRE12~ = !WE1~ & !BE2~;

   !WRE03~ = !WE0~ & !BE3~;

   !WRE13~ = !WE1~ & !BE3~;


end SC_MODE_DRAM_CTRL_12;
```

240799-49

2

```
module          SC_MODE_DRAM_CTRL_15        flag '-r4'

title    'DRAM CONTROLLER - PLD 15, INTEL CORPORATION'
" This PLD combines ready signals
" Implemented with the Intel 85C220 EPLD

     SC15     device       'E0320';

     x        =      .X.;          " ABEL 'don't care' symbol
     c        =      .C.;          " ABEL 'clocking input' symbol

" Inputs

     MEMCS~           pin    1;     "Memory Chip Select
     JRDY~            pin    2;     "
     MRDY~            pin    3;     "Memory Ready for Write Cycles
     BRDY~            pin    4;     "Processor BRDY for Read Cycles
     ALD              pin    5;     "Address latch disable
     CKEN~            pin    6;     "KEN# output of 485Turbocache
     SKEN~            pin    7;     "KEN# input to 485Turbocache
     BRDYO~           pin    8;     "Burst Ready Out from 485Turbocache
     M~               pin    9;     "Miss Indicator
     CIP~             pin   11;     "Cycle OK

" Output

     WEN~             pin   12;     "Write enable for write latches
     RDY~             pin   13;     "Processor RDY input
     MRDYCS~          pin   14;     "
     MALD~            pin   15;     "Modified ALD for FF's
     dum10            pin   16;     "
     PBRDY~           pin   17;     "Processor BRDY# input
     KEN~             pin   18;     "Processor KEN# input
     DRDY~            pin   19;     "


equations

       !MALD~ = (!MEMCS~ & !ALD);

       !RDY~ = (!MRDY~ & M~ & !MEMCS~) # !JRDY~;

       !MRDYCS~ = (!MRDY~ & M~ & !MEMCS~);

       !WEN~ = !CIP~ & M~;

       !DRDY~ = !BRDY~ # !MRDYCS~;

       KEN~ = SKEN~ & CKEN~;

       PBRDY~ = BRDY~ & BRDYO~;

end SC_MODE_DRAM_CTRL_15;
```

240799-50

```
module          SC_MODE_DRAM_CTRL_16        flag '-r4'

title   'DRAM CONTROLLER - PLD 16, INTEL CORPORATION'
" This PLD generates the chip select outputs.
" Implemented with the Intel 85C220 EPLD

    SC16   device       'E0320';

    x       =      .X.;          " ABEL 'don't care' symbol
    c       =      .C.;          " ABEL 'clocking input' symbol

" Inputs

    O1              pin   1;     "Decode Outputs of Intel 85C508 decoder/latch
    O2              pin   2;     "
    O3              pin   3;     "
    O4              pin   4;     "
    O5              pin   5;     "
    BRDYO~          pin   6;     "Burst Ready Output of 485Turbocache
    CDIS            pin   7;     "Jumper select, disables cache enable of 485
    M_IO            pin   8;     "Processor M/IO# output
    D_C             pin   9;     "Processor D/C# output

" Output

    dum7            pin  12;     "
    KSEL~           pin  13;     "Active to enable cacheing in 485
    MEMCS~          pin  14;     "Memory Chip Select
    CS54~           pin  15;     "
    CS510~          pin  16;     "
    JMCS~           pin  17;     "
    EPCS~           pin  18;     "
    RAMCS~          pin  19;     "

equations

      RAMCS~ = MEMCS~ # !MEMCS~;

      !MEMCS~ = (!O1 & M_IO & BRDYO~) # (O2 & !O4 & M_IO & BRDYO~);

      !EPCS~ = (!O2 # !O3) & M_IO;

      !CS510~ =!O5 & !M_IO & D_C;

      !CS54~ =  O5 & !M_IO & D_C;

      !JMCS~ = (!EPCS~ # !CS510~ # !CS54~);

    !KSEL~ = !MEMCS~ & !CDIS;

end SC_MODE_DRAM_CTRL_16;
```

240799-51

2

```
module          SC_MODE_DRAM_CTRL_17        flag '-r4'

title   'DRAM CONTROLLER - PLD 17, INTEL CORPORATION'
" This PLD generates A0 for Bank 1
" Implemented with the Intel 85C224 EPLD

    SC17    device      'E224';

    x       =       .X.;        " ABEL 'don't care' symbol
    c       =       .C.;        " ABEL 'clocking input' symbol

" Inputs

    CLK             pin   1;    "i486 CPU input CLK"
    BRDY~           pin   2;    "Burst Ready
    CIP~            pin   3;    "Cycle OK
    MEMCS~          pin   4;    "Memory Chip Select
    LA313           pin   5;    "Latched A3 and A13
    DATASEL         pin   6;    "Bank Select for Reads
    RAS~            pin   7;    "Row address strobe
    LW_R            pin   8;    "CPU W/R latched~
    RESET           pin   9;    "System Reset
    BLAST~          pin  10;    "CPU BLAST~ output
    A3              pin  11;    "CPU A3 line
    ALD             pin  14;    "Address Latch disable
    dum1            pin  15;
    WE1~            pin  22;    "Write enable
    dum2            pin  23;

" Output

    B10MA0          pin  21;    "Bank 1 A0
    B1A             pin  20;
    CS0~            pin  19;
    dun             pin  18;
    dum             pin  17;
    B11MA0          pin  16;    "Bank 1 A0

state_diagram [B1A, CS0~]

    state [1, 1]: if RESET then [1, 1] else
                  if CIP~ & !ALD & !A3 then [0, 1] else
                  if !CIP~ & !ALD & !A3 then [0, 1] else
                  if !CIP~ & !LW_R & !MEMCS~ & WE1~ then [1, 0] else [1, 1];

    state [0, 1]: if RESET then [1, 1] else
                  if CIP~ & !ALD & A3 then [1, 1] else
                  if !CIP~ & !ALD & A3 then [1, 1] else
                  if !CIP~ & !LW_R & !MEMCS~ & WE1~ then [0, 0] else [0, 1];


    state [1, 0]: if RESET #  (!BRDY~ & !BLAST~) then [1, 1] else
                  if !BRDY~ & !DATASEL then [0, 0] else [1, 0];

    state [0, 0]: if RESET #  (!BRDY~ & !BLAST~) then [1, 1] else
                  if !BRDY~ & !DATASEL then [1, 0] else [0, 0];

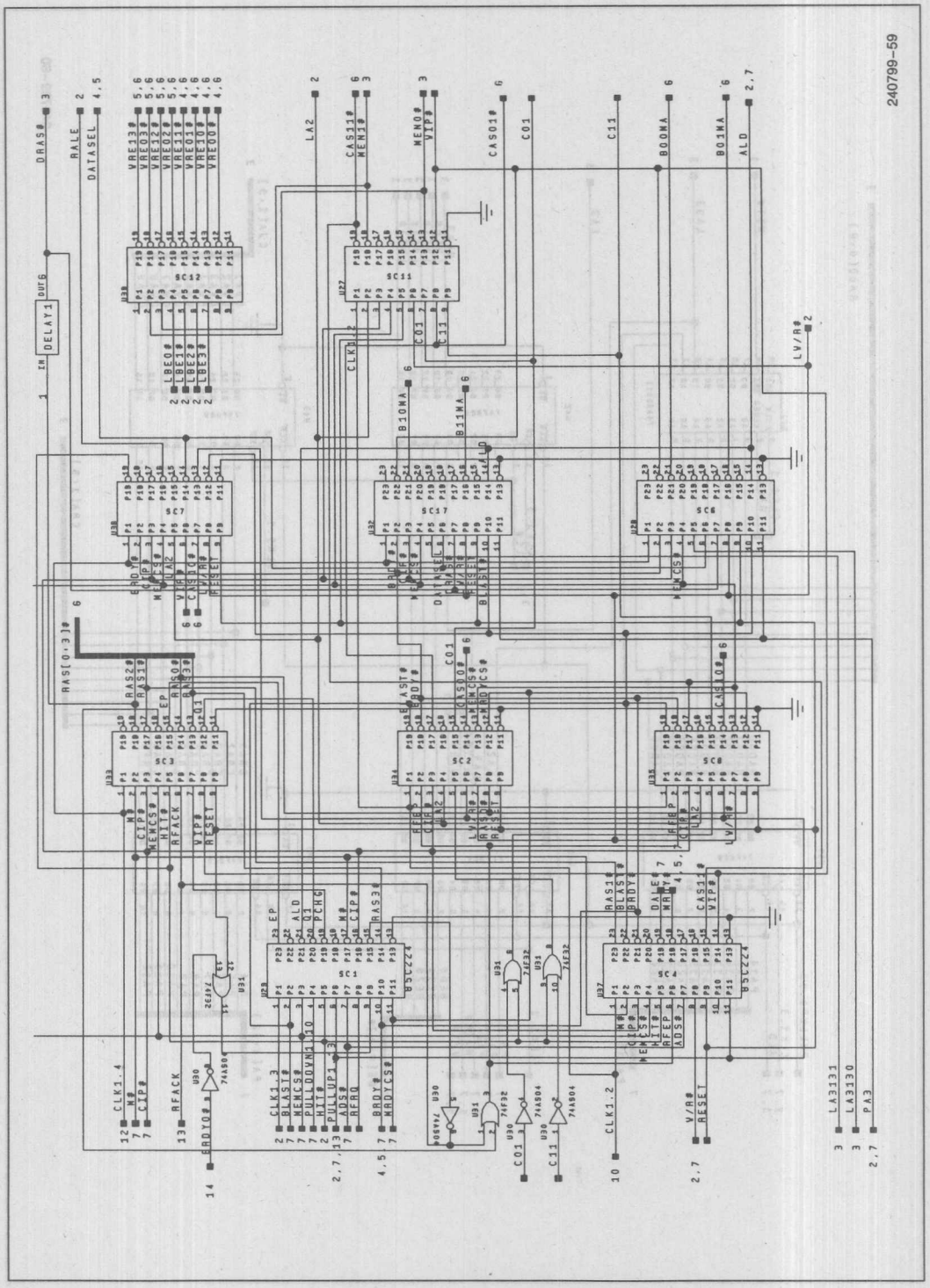
equations

!B10MA0 = !WE1~ & !LA313 # WE1~ & RAS~ & !LA313 # WE1~ & !RAS~ & !B1A;

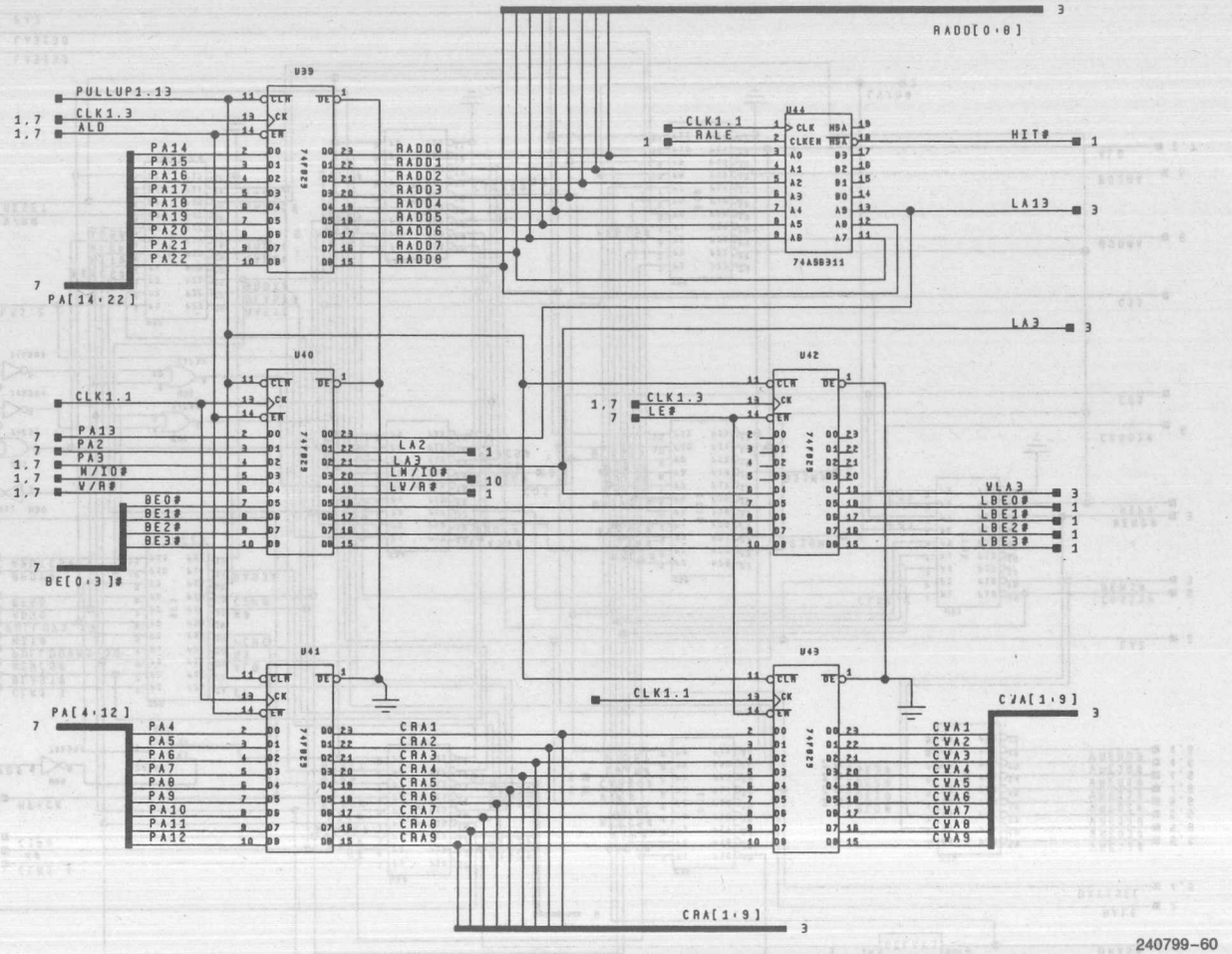
!B11MA0 = !WE1~ & !LA313 # WE1~ & RAS~ & !LA313 # WE1~ & !RAS~ & !B1A;

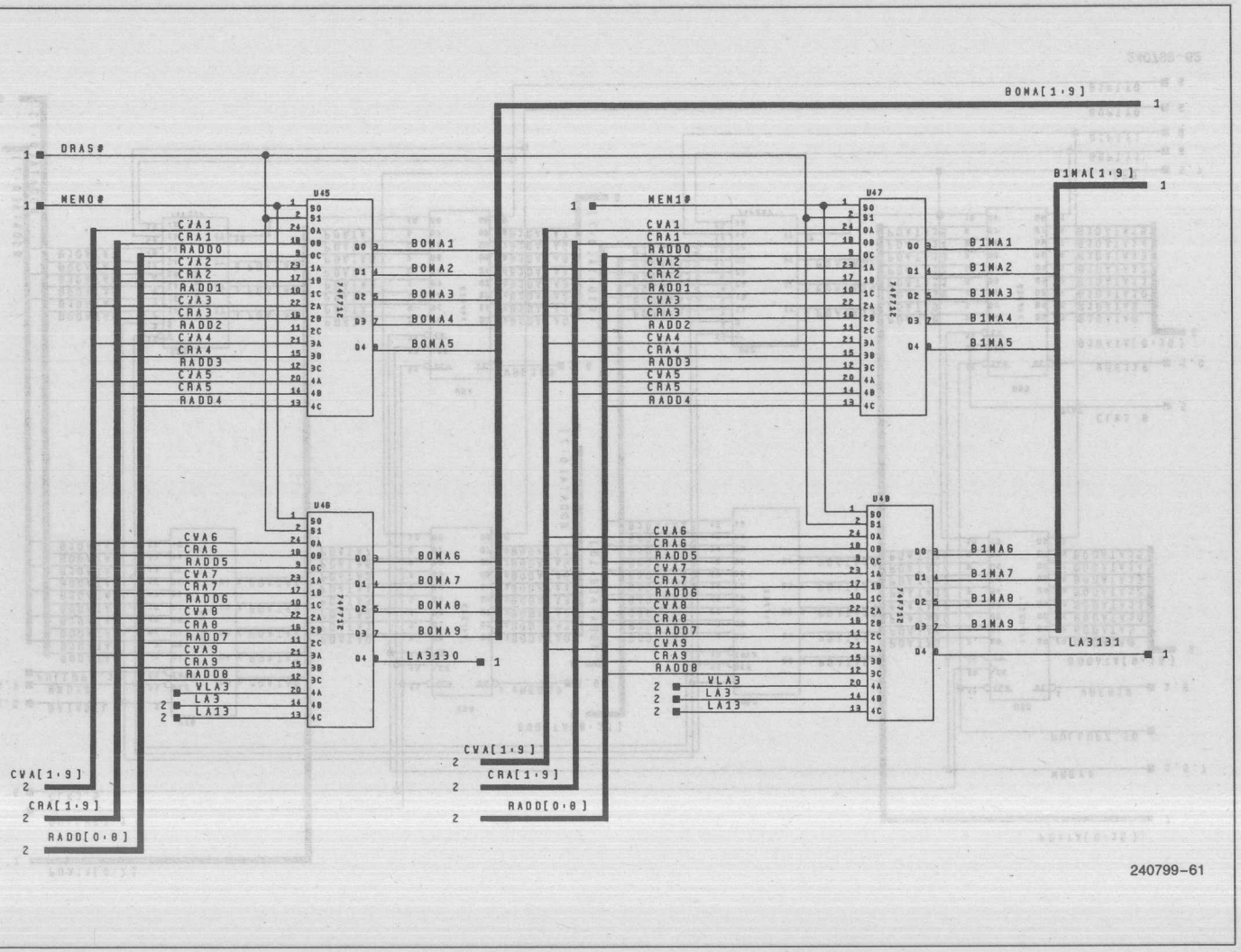
end SC_MODE_DRAM_CTRL_17;
```

240799-52

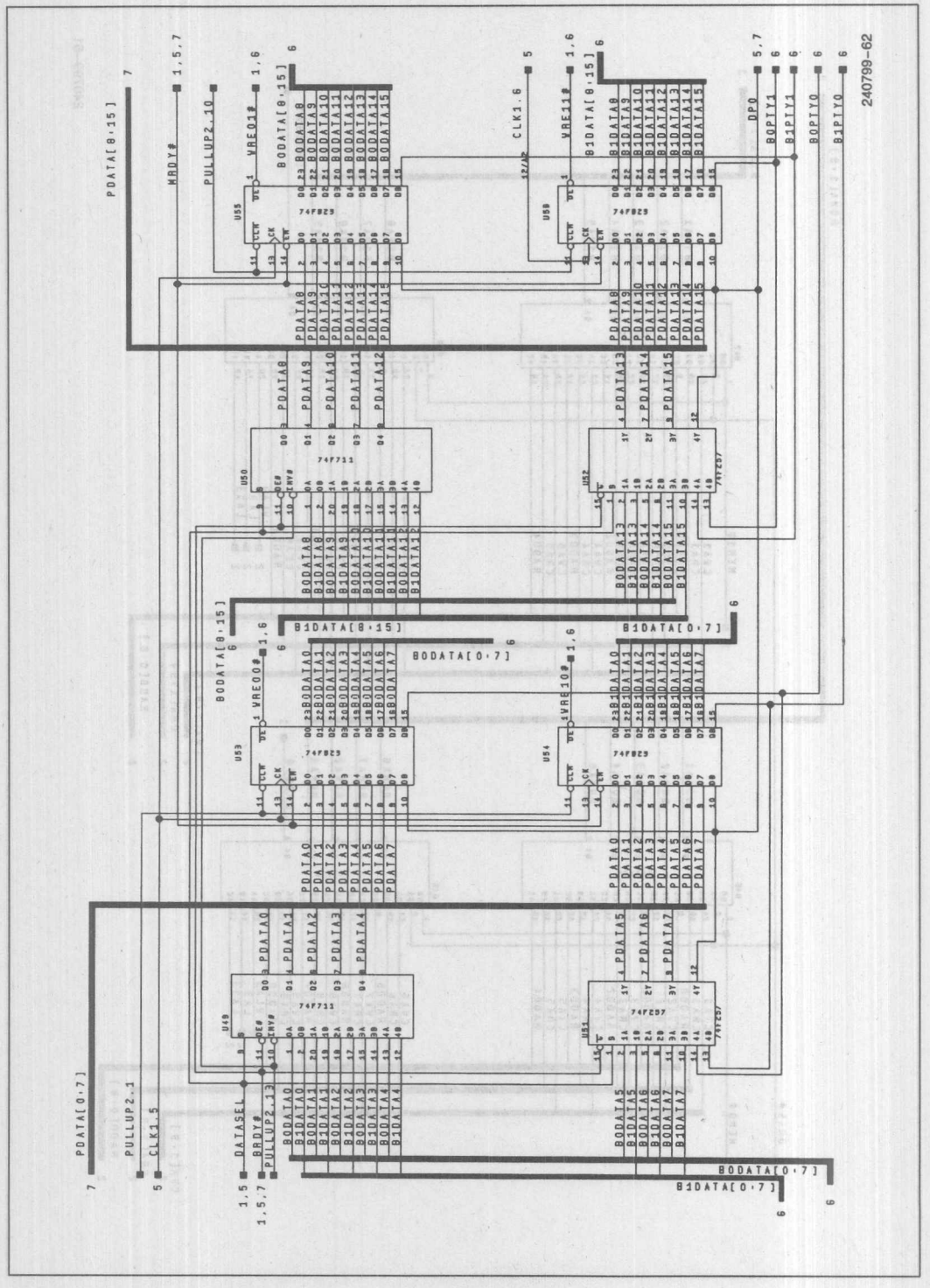

240799-59
DRAS#
RALE
DATASEL
DELAY1
VRE13#
VRE03#
VRE12#
VRE02#
VRE11#
VRE01#
VRE10#
VRE00#
SC12
SC11
SC7
SC17
SC6
SC3
SC2
SC8
SC1
SC4
LBE0#
LBE1#
LBE2#
LBE3#
LA2
CAS11#
MEM1#
MEM0#
VIP#
CAS01#
C01
C11
BOOMA
B01MA
ALD
LV/R#
RAS[0..3]#
RAS2#
RAS3#
RAS0#
RAS1#
BLAST#
BOOT#
MEMCS0#
MEMCS1#
RESET
CLK1.4
M#
C1P#
RFACK
BRDY0#
CLK1.3
BLAST#
MEMCS0#
HITO#
PULLUP1
ADS#
RFRQ
BRDY#
MRDYCS#
CLK1.2
C01
C11
LA3133
LA3130
PA3

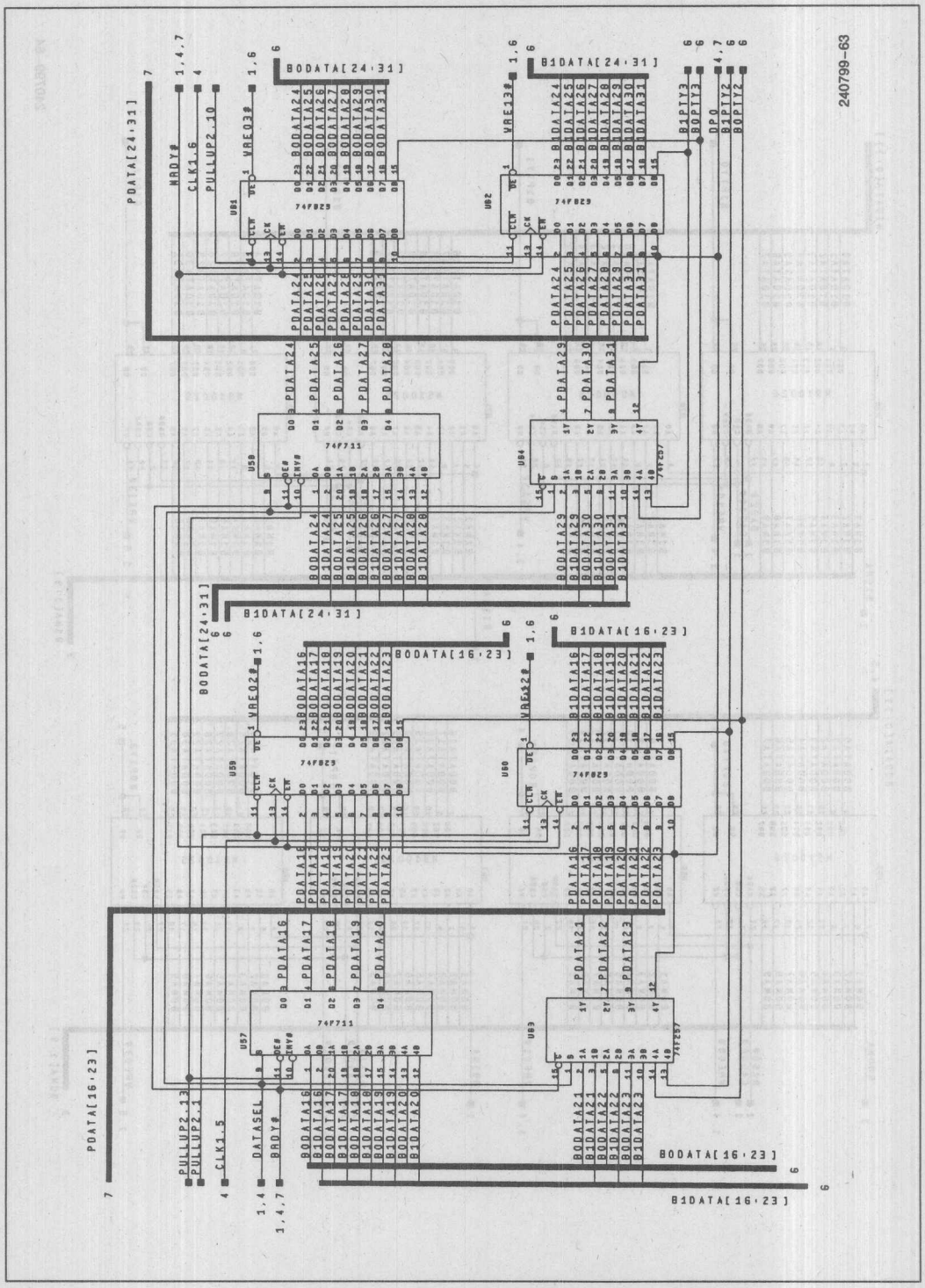
2

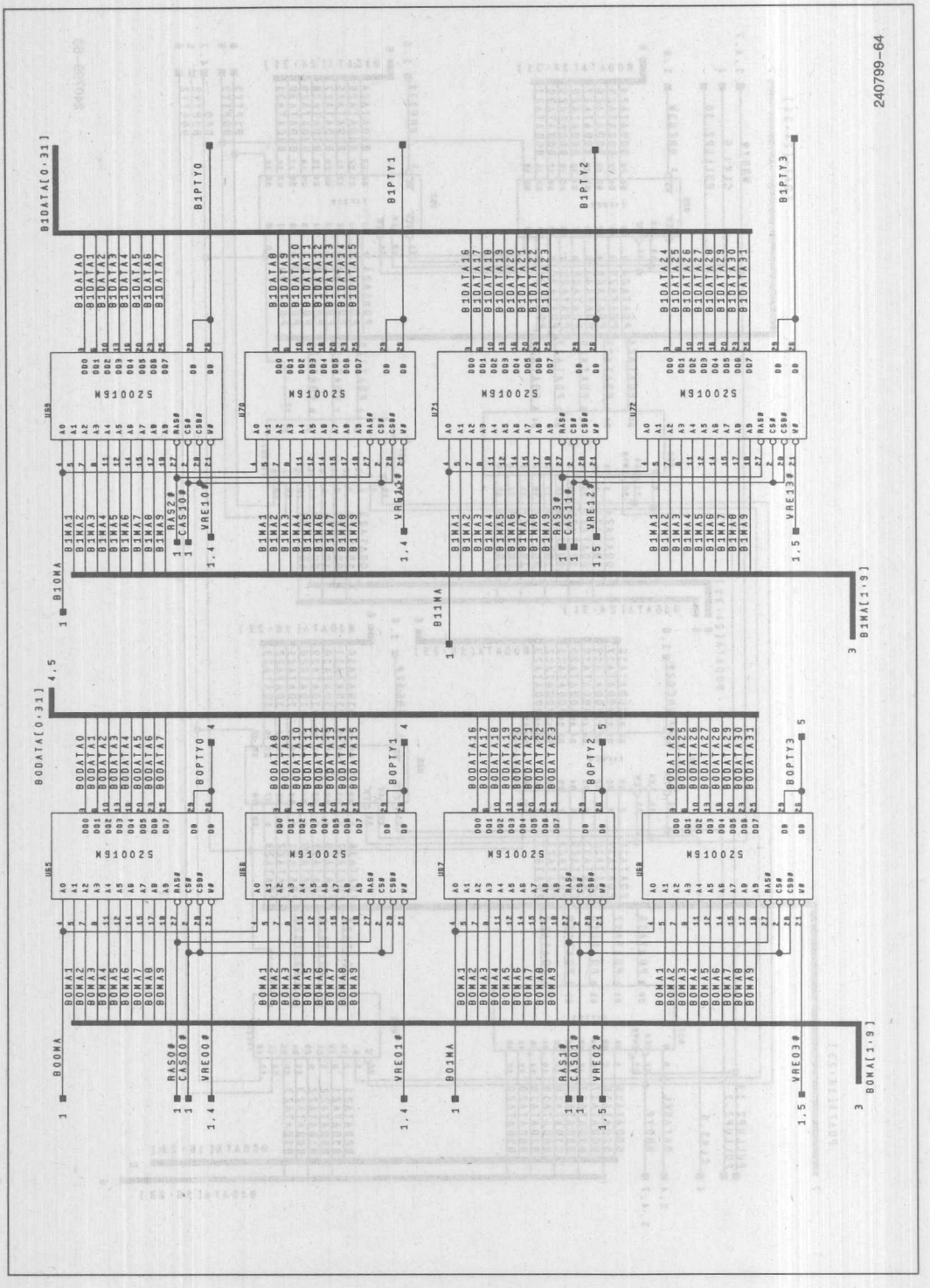

RADD[0:8]
3
U39
PULLUP1.13
CLK1.3
1,7
ALD
1,7
PA14
PA15
PA16
PA17
PA18
PA19
PA20
PA21
PA22
7
PA[14:22]
RADD0
RADD1
RADD2
RADD3
RADD4
RADD5
RADD6
RADD7
RADD8
U44
CLK1.1
1
RALE
74ASB311
HIT#
1
LA13
3
LA3
3
U40
CLK1.1
7
PA13
7
PA2
1,7
PA3
1,7
W/IO#
1,7
W/R#
BE0#
BE1#
BE2#
BE3#
7
BE[0:3]#
LA2
1
LA3
LW/IO#
10
LW/R#
1
U42
1,7
CLK1.3
7
LE#
VLA3
3
LBE0#
1
LBE1#
1
LBE2#
1
LBE3#
1
U41
PA[4:12]
7
PA4
PA5
PA6
PA7
PA8
PA9
PA10
PA11
PA12
CRA1
CRA2
CRA3
CRA4
CRA5
CRA6
CRA7
CRA8
CRA9
U43
CLK1.1
CVA[1:9]
3
CVA1
CVA2
CVA3
CVA4
CVA5
CVA6
CVA7
CVA8
CVA9
CRA[1:9]
3
240799-60

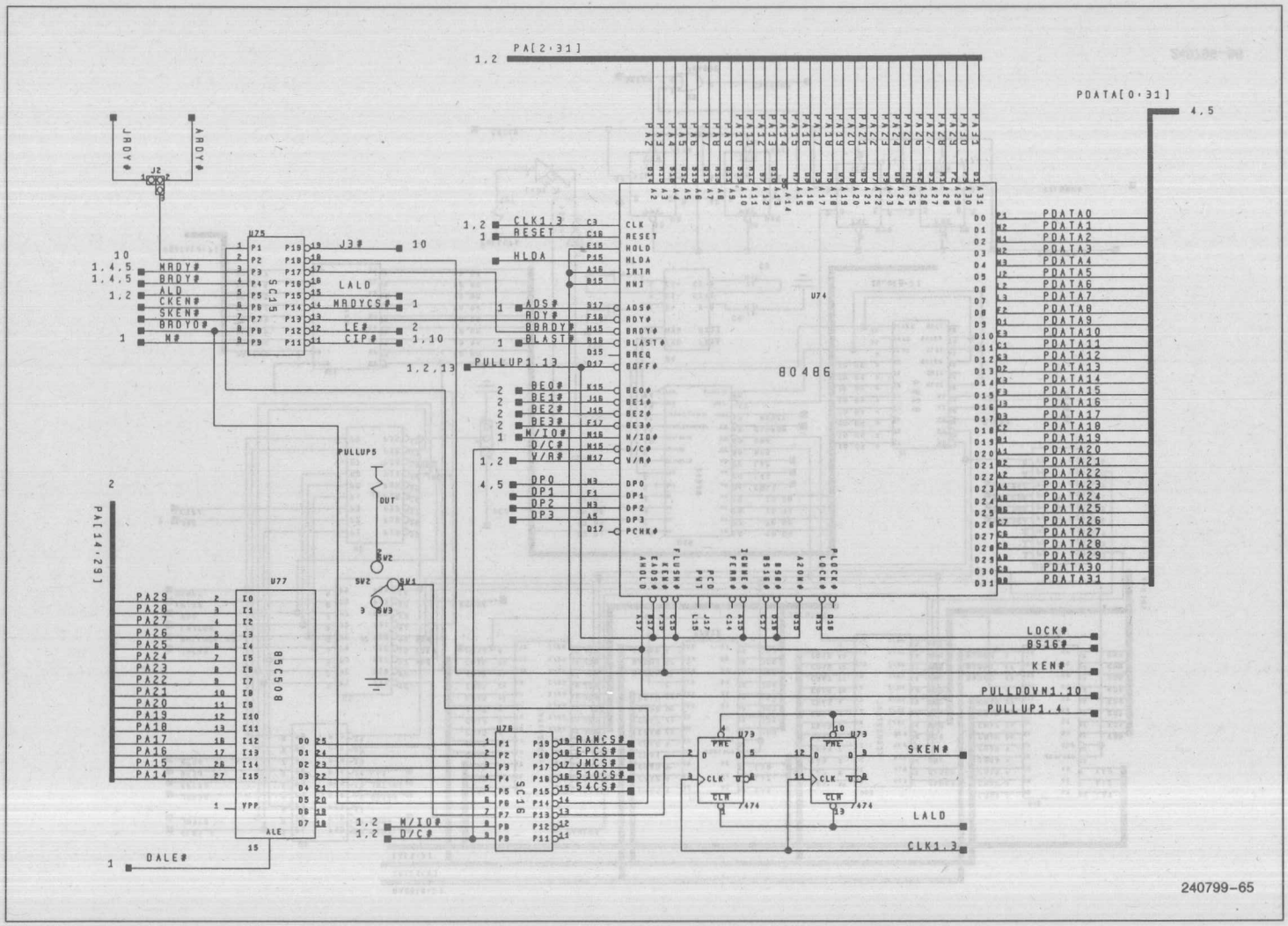

BOMA[1:9]
DRAS#
MEN0#
MEN1#
U45
U46
U47
U48
74F512
CVA1
CRA1
RADD0
CVA2
CRA2
RADD1
CVA3
CRA3
RADD2
CVA4
CRA4
RADD3
CVA5
CRA5
RADD4
CVA6
CRA6
RADD5
CVA7
CRA7
RADD6
CVA8
CRA8
RADD7
CVA9
CRA9
RADD8
WLA3
LA3
LA13
BOMA1
BOMA2
BOMA3
BOMA4
BOMA5
BOMA6
BOMA7
BOMA8
BOMA9
LA3130
B1MA[1:9]
B1MA1
B1MA2
B1MA3
B1MA4
B1MA5
B1MA6
B1MA7
B1MA8
B1MA9
LA3131
CVA[1:9]
CRA[1:9]
RADD[0:8]

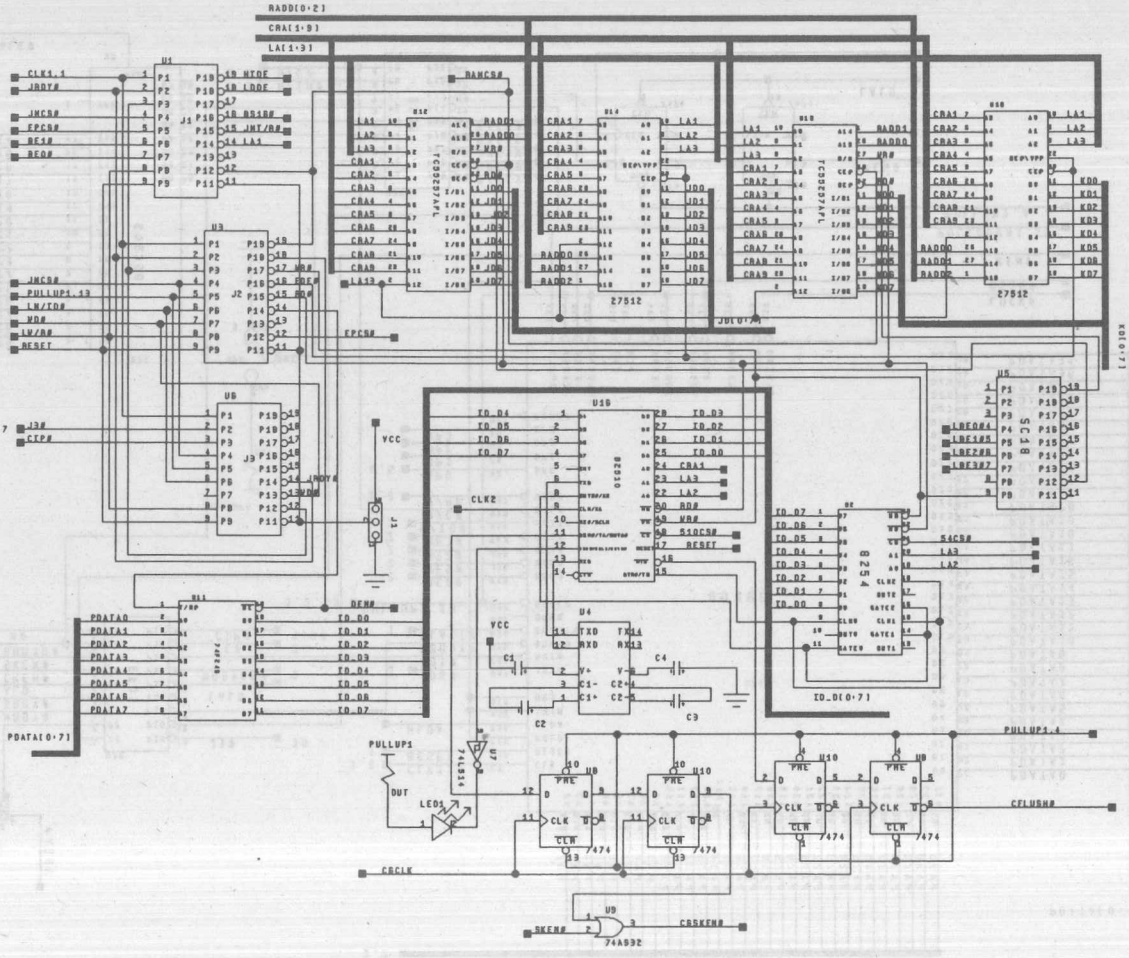
240799-61

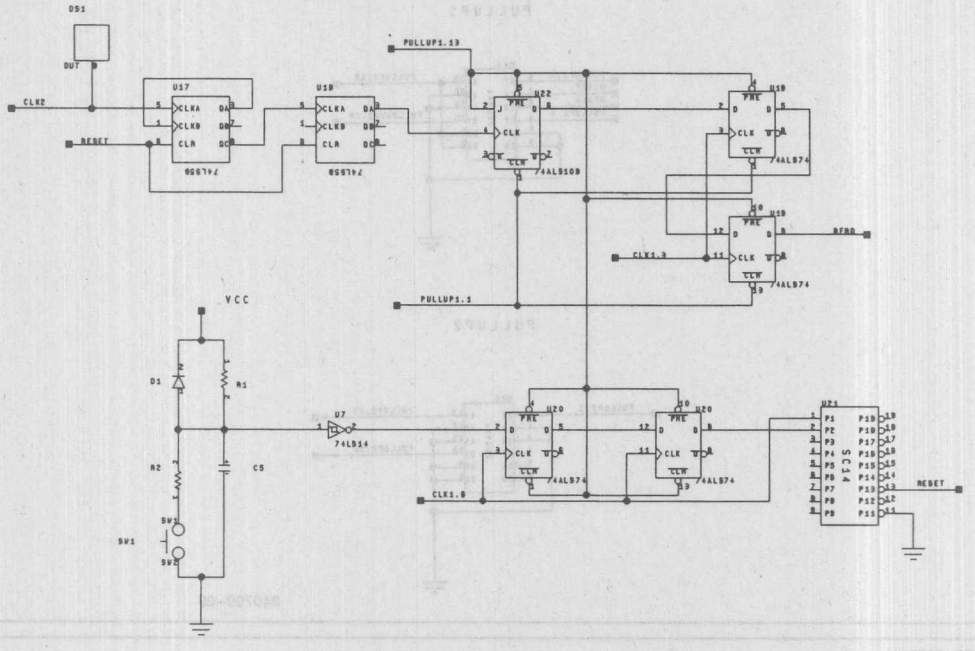

240799-62


240799-63
PDATA[24:31]
BODATA[24:31]
B1DATA[24:31]
PDATA[16:23]
BODATA[16:23]
B1DATA[16:23]
74F823
74F731
74F257
NRDY#
CLK1.6
PULLUP2.10
VRE03#
VRE13#
VRE02#
VRE12#
PULLUP2.13
PULLUP2.1
CLK1.5
DATASEL
BRDY#
B1PIT3
B0PIT3
DP0
B1PIT2
B0PIT2

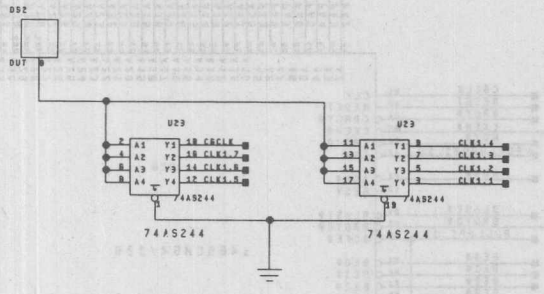

240799-64


PA[2:31]
PDATA[0:31]
80486
U74
U75
SCS
U77
85C508
U78
U73
7474
CLK1.3
RESET
HLDA
ADS#
RDY#
BRDY#
BLAST#
PULLUP1.13
BE0#
BE1#
BE2#
BE3#
M/IO#
D/C#
W/R#
DP0
DP1
DP2
DP3
PCHK#
PULLUP5
PA[14:29]
MRDY#
BRDY#
ALD
CKEN#
SKEN#
BRDYO#
M#
J3#
LALD
MRDYCS#
LE#
CIP#
DALE#
RAMCS#
EPCS#
JMCS#
512CS#
54CS#
LOCK#
BS16#
KEN#
PULLDOWN1.10
PULLUP1.4
SKEN#
LALD
CLK1.3
240799-65


RADD[0:2]
CRA[1:9]
LA[1:3]
RAMCS#
27512
74LS257A
74LS245
SC18
82C54
82510
74F245
74LS04
74LS74
74A32
PULLUP1
LED1
VCC
CLK2
CRCLK
FPCS#
PDATA[0:7]
ID_D[0:7]
CFLUSH#
240799-66



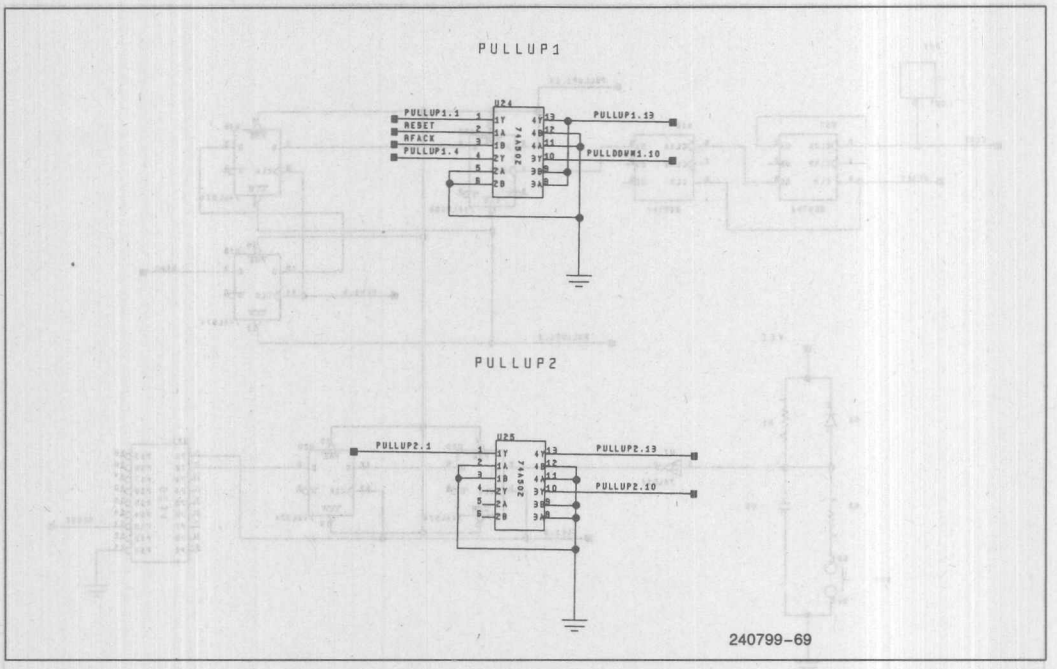
240799-67

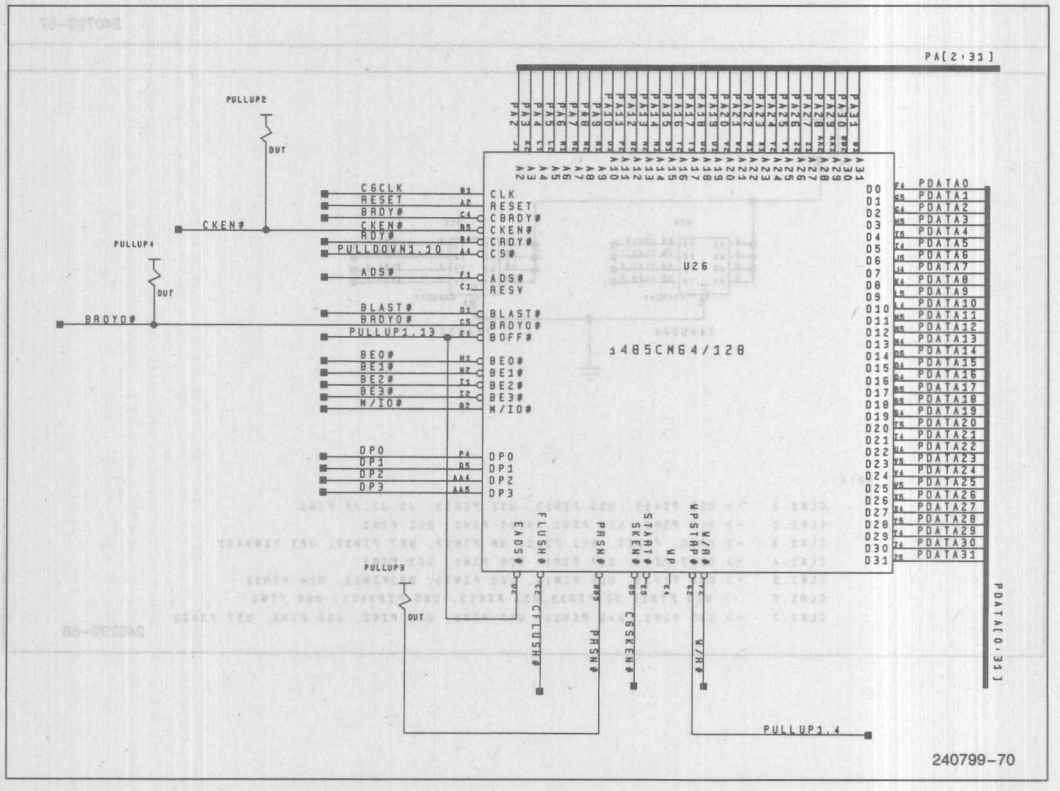


NOTE:

CLK1.1 -> U10 PIN13, U13 PIN13, U12 PIN13, J1,J2,J3 PIN1
CLK1.2 -> SC4 PIN1, SC8 PIN1, SC11 PIN1, SC2 PIN1
CLK1.3 -> PROC. PIN3, SC1 PIN1, U8 PIN13, U87 PIN13, U83 PIN3&11
CLK1.4 -> SC17 PIN1, SC7 PIN1, SC6 PIN1, SC3 PIN1
CLK1.5 -> U30 PIN13, U25 PIN13, U29 PIN13, U21PIN13, U24 PIN13
CLK1.6 -> U20 PIN13,U27PIN13,U33 PIN13, U85 PIN3&11, U86 PIN1
CLK1.7 -> U45 PIN1, U46 PIN18, U47 PIN1, U49 PIN2, U56 PIN6, U57 PIN23

240799-68


PULLUP1
PULLUP1.1
RESET
AFACK
PULLUP1.4
U24
74ALS02
PULLUP1.13
PULLDOWN1.10
PULLUP2
PULLUP2.1
U25
74ALS02
PULLUP2.13
PULLUP2.10
240799-69


PA[2:31]
PULLUP2
PULLUP4
PULLUP3
CKEN#
BRDY0#
C6CLK
RESET
BRDY#
CKEN#
RDY#
PULLDOWN1.10
ADS#
BLAST#
BRDY0#
PULLUP1.13
BE0#
BE1#
BE2#
BE3#
M/IO#
DP0
DP1
DP2
DP3
U26
i486CM64/128
CLK
RESET
CBRDY#
CCKEN#
CRDY#
CS#
ADS#
RESV
BLAST#
BRDY0#
BOFF#
BE0#
BE1#
BE2#
BE3#
M/IO#
DP0
DP1
DP2
DP3
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15
D16
D17
D18
D19
D20
D21
D22
D23
D24
D25
D26
D27
D28
D29
D30
D31
PDATA0
PDATA1
PDATA2
PDATA3
PDATA4
PDATA5
PDATA6
PDATA7
PDATA8
PDATA9
PDATA10
PDATA11
PDATA12
PDATA13
PDATA14
PDATA15
PDATA16
PDATA17
PDATA18
PDATA19
PDATA20
PDATA21
PDATA22
PDATA23
PDATA24
PDATA25
PDATA26
PDATA27
PDATA28
PDATA29
PDATA30
PDATA31
PDATA[0:31]
EADS#
FLUSH#
PRDY#
PREQ#
START#
WP
WSTRB#
M/R#
CFLUSH#
CSKEN#
V/R#
PULLUP1.4
240799-70

intel®

AP-453

APPLICATION NOTE

Clock Design in 50 MHz Intel486™ Systems

2

RICH JOLLY
INTEL TECHNICAL MARKETING

October 1991

PRELIMINARY
Order Number: 241082-002

CONTENTS

PAGE

1.0 INTRODUCTION 2-933

2.0 CLOCK DISTRIBUTION IN SYSTEMS 2-934

3.0 DESIGNING A SYSTEM CLOCK ... 2-938

3.1 Design Example 2-938

3.2 Choosing a Clock Driver 2-940

3.3 Clock Signal Layout 2-942

3.4 Termination 2-945

3.5 Simulation 2-945

4.0 SUMMARY 2-947

5.0 REFERENCES 2-947

APPENDIX A—Methods to Measure the Output Impedance of a Clock Driver 2-949

FIGURES

Figure 1-1 The simple representation for components follows the lumped load approach. 2-933

Figure 1-2 With faster edge rates, the interconnection between components must now be modeled as a transmission line. 2-934

Figure 2-1 System Design with Bus Frequency Hierarchy 2-934

Figure 2-2 Traditional Clock Routing Scheme 2-935

Figure 2-3 Local Generation of Asynchronous Clocks 2-935

Figure 2-4 Local Generation of Synchronous Clocks 2-936

Figure 2-5 Sources of Clock Skew 2-937

Figure 3-1 A CPU Module with a 50 MHz Intel486™ DX Processor and 256K Byte Second Level Cache 2-938

Figure 3-2 Clock Requirement for the 50 MHz Intel486 DX CPU-Cache Chip Set and Module 2-938

CONTENTS

PAGE

Figure 3-3 Model of the Clock Distribution to the Intel486 DX CPU-Cache Subsystem 2-939

Figure 3-4 Comparison of a PLL and Buffered Clock Circuit 2-940

Figure 3-5 Design Example for the Generation of Multiple Synchronized 2x Clocks ... 2-942

Figure 3-6 Several Methods of Routing Clock Signals 2-943

Figure 3-7 Routing of the Clocks on the Intel486 DX CPU-Cache Module 2-944

Figure 3-8 Simple I/O Model of a Clock Driver 2-945

Figure 3-9 SPICE Simulation of Clock Waveforms at the Inputs to the Intel486™, 82495DX and 82490DX Components 2-946

Figure A-1 Extraction of Output Driver Pull-Up Impedance with the I-V Method 2-949

Figure A-2 Test Circuit and Response for the Derating Curve Method of Extracting the Output Driver's Impedance 2-949

Figure A-3 The derating curve of an output driver. The slope is proportional to R. 2-949

Figure A-4 Test Circuit Used to Extract the Output Impedance with the Open Transmission Line Method 2-950

Figure A-5 Output at Node A of a Driver into an Open Transmission Line 2-950

TABLES

Table 3-1 Table of Clock Drivers 2-941

1.0 INTRODUCTION

With the introduction of 33 MHz Intel386™ microprocessors, desk top system designers needed to account for a previously limited phenomena, high speed signal effects. As outlined in the Intel Application Note, *33 MHz Intel386™ System Design Considerations* [1], as edge rates become a more significant portion of the clock cycle transmission line effects take hold. When extending system designs to 50 MHz the effects intensify themselves. What were once minor reflections become large spikes. Driver characteristics, signal routing, load characteristics take on greater significance and electrical simulations (such as SPICE) becomes essential. One of the most critical signals is the clock. Not only is signal quality a key issue, but the skew between different instantiations of the clock can cause serious system problems. In this application note, the key issues in the design of a 50 MHz clock for an Intel486 system will be discussed. These will be illustrated with the clock design for a CPU cache subsystem using the 82495DX cache controller and 82490DX Cache SRAM in an Intel486™ DX CPU-Cache module.

The Intel486 DX CPU-Cache chip set forms the core of a CPU-second level cache module. Along with a memory bus controller, this chip set provides a CPU-like interface for many types of memory buses. The Intel486 DX CPU-Cache chip set is described in the following Intel literature [2, 3].

Previous designs have viewed signal traces as simple lumped components connected by direct wiring as shown in Figure 1-1. However, we know that at very high frequencies, long traces behave like transmission lines as shown in Figure 1-2. When does the simple lumped load model break down? Simply put, you must consider transmission line effects when the signal propagation time becomes comparable to the rise or fall times of the signal. This can be restated with simple rules of thumb either in the frequency domain as:

interconnect length > 1/15th of the wavelength

or in time domain as:

signal transit time > 1/8th the signal rise or fall time

Thus, the designer must know the signal rise and fall times, the characteristic impedance of the interconnect, and the source and load impedances. Several excellent treatments of these subjects exist in the open literature [refs] so it will not be repeated here. For signals with 1 ns–3 ns rise and fall times (10% to 90%) the maximum length of interconnect before transmission line effects are significant range from about 2.1 inches to 9.1 inches for 50Ω traces.

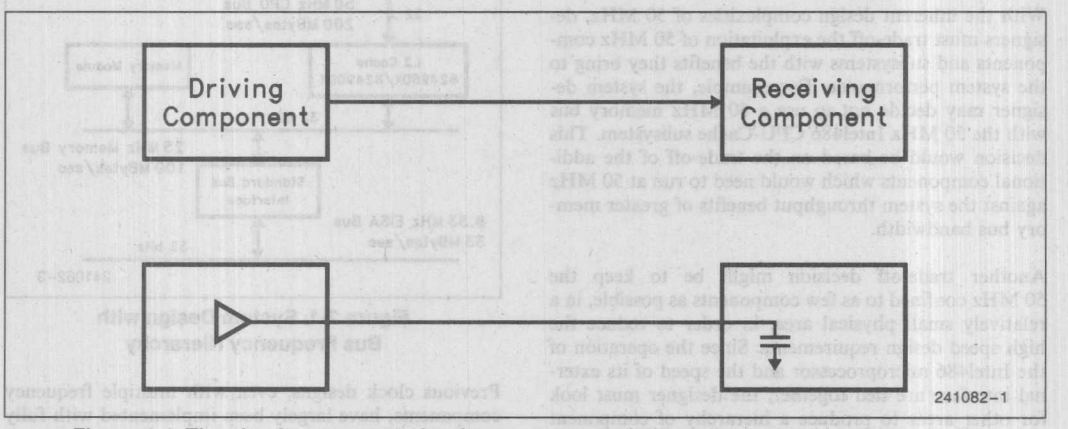


Figure 1-1. The simple representation for components follows the lumped load approach. Connections between components are simply modeled as a wire.

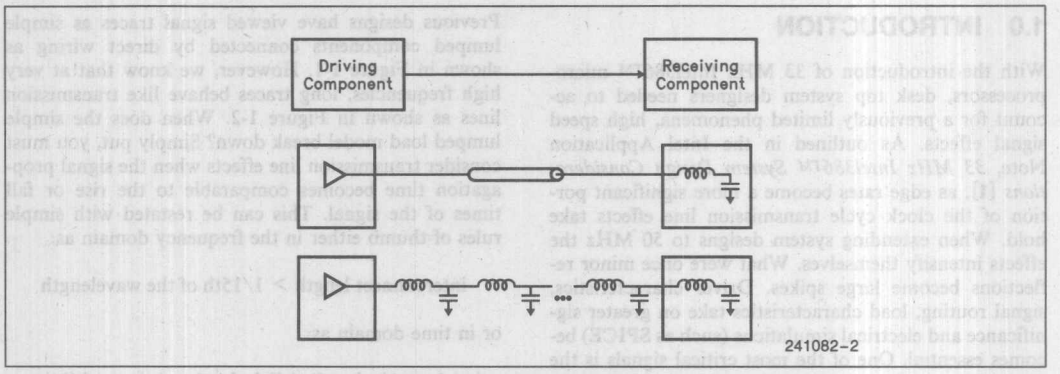


Figure 1-2. With faster edge rates, the interconnection between components must now be modeled as a transmission line.

2.0 CLOCK DISTRIBUTION IN SYSTEMS

The primary goals of a system clock distribution is to deliver a clock signal to each component's input pins which meets the system's requirements for: component to component skew; acceptable waveform shape (rise time, fall time, overshoot, undershoot, high and low times, duty cycle); and stability (cycle to cycle). With the complexities involved in distributing a system clock at these high frequencies, system clock design plays a major role in the overall system architecture.

With the inherent design complexities of 50 MHz, designers must trade-off the exploitation of 50 MHz components and subsystems with the benefits they bring to the system performance. For example, the system designer may decide not to use a 50 MHz memory bus with the 50 MHz Intel486 CPU-Cache subsystem. This decision would be based on the trade-off of the additional components which would need to run at 50 MHz against the system throughput benefits of greater memory bus bandwidth.

Another trade-off decision might be to keep the 50 MHz confined to as few components as possible, in a relatively small physical area, in order to reduce the high speed design requirements. Since the operation of the Intel486 microprocessor and the speed of its external interface are tied together, the designer must look for other areas to produce a hierarchy of component speeds.

An important benefit of a look through second level cache has always been a level of isolation between the CPU and memory buses. However, in the past, this benefit has mainly been realized as reduced transactions based on cache hits. Now, this second level cache provides a *frequency isolation* function in addition to transaction isolation. Figure 2-1 below shows a possible system implementation with a hierarchy of buses ranging from the high frequency, 50 MHz, CPU bus down to the lower speed, 8.33 MHz EISA bus. This type of design keeps the critical 50 MHz signals easily confined to a small physical area—paramount to reducing transmission line effects.

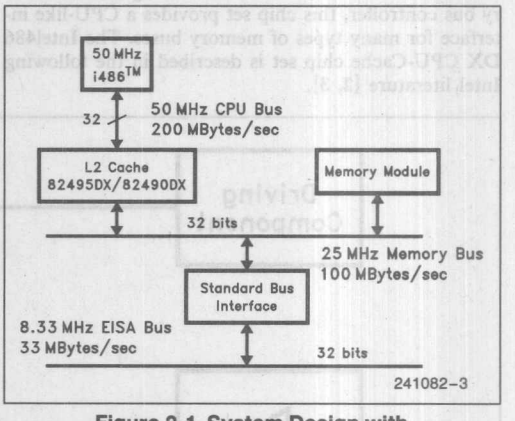


Figure 2-1. System Design with Bus Frequency Hierarchy

Previous clock designs, even with multiple frequency components, have largely been implemented with fully synchronous circuitry. While synchronous designs typically have the highest performance and simplify the logical design of the system, at 50 MHz they will test the designer's ability to route and generate acceptable quality clocks. Figures 2-2 through 2-4 show three possible methods of implementing clocking schemes for the triple frequency system design shown in Figure 2-1.

Figure 2-2 shows the traditional method of clock generation. A 50 MHz oscillator (a 100 MHz differential PECL oscillator may be used with some clock driver chips) feeds a clock generation circuit to produce a number of 50 MHz clock signals. These are routed to the 50 MHz components (Intel486 DX CPU-Cache subsystem and memory bus controller) and the 25 MHz clock generation circuitry. Here multiple 25 MHz clocks are generated synchronously (but delayed) from the master 50 MHz clock. Some of the newer clock generation chips also contain subharmonic outputs which could also produce the 25 MHz clocks. These 25 MHz clocks are routed to the 25 MHz components (the memory module and bus interface unit) and the 8.33 MHz clock generation circuits. In this scheme, the 8.33 MHz clock will also be synchronous to the 25 MHz and 50 MHz clocks, but delayed from both. Note, however, that the delay becomes less critical with the lower frequency signals.

Figure 2-2 also shows a variant of this scheme where the oscillator output is routed to additional clock generation circuits rather than routing the 50 MHz clocks. This may have an advantage depending on the oscillator design and the logic levels used to route the oscillator output.

A second method of clock distribution relies on the generation of a 25 MHz master signal and its routing to 25 MHz components and the local generation of asynchronous 50 MHz clocks as shown in Figure 2-3. The advantage of this scheme is the simplified routing of high speed signals if multiple 50 MHz clocks are required (for example for a second CPU module in a multi-processing system). The disadvantages are increased cost due to multiple oscillators and the reduced performance of a CPU/memory bus that is not fully synchronous.

2

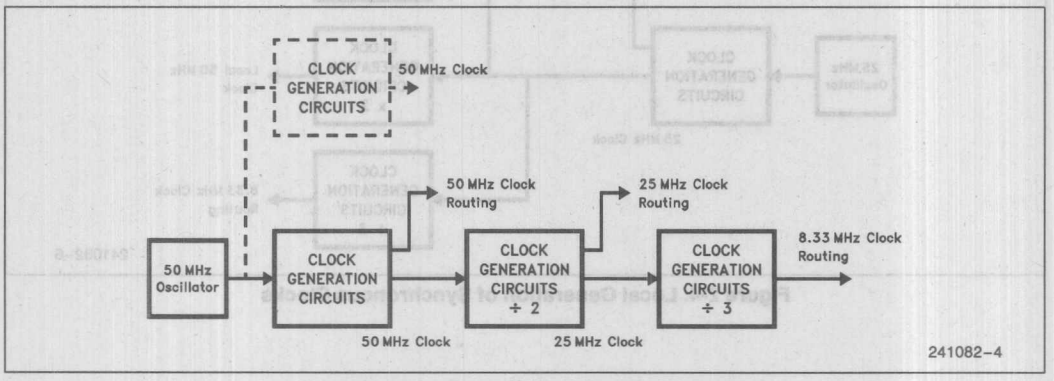


Figure 2-2. Traditional Clock Routing Scheme

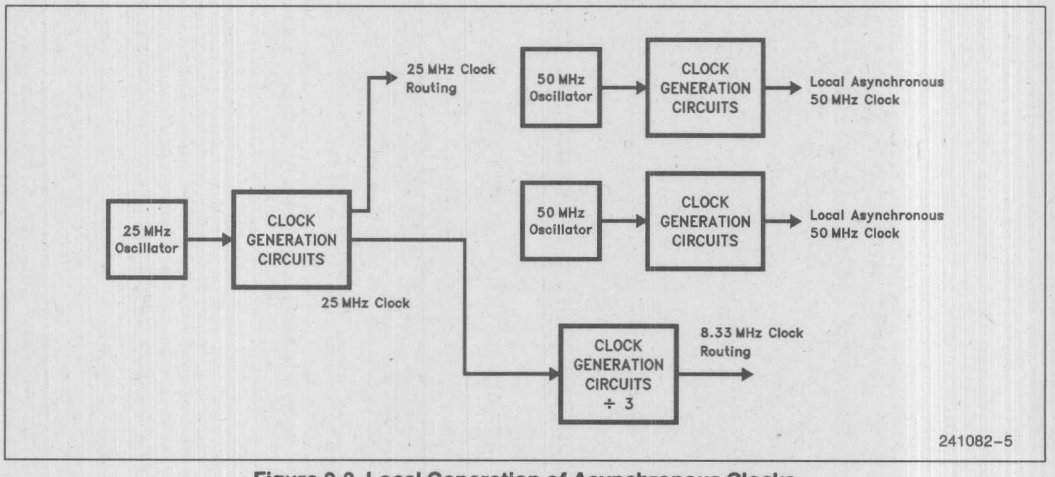


Figure 2-3. Local Generation of Asynchronous Clocks

A new type of clock driver circuit provides a third method of clock generation. This new class of clock chip has an internal phase lock loop circuit which synchronizes the output clock to the input clock. Also, since the chip's internal clock is running much faster than the input or output clock, multiple (usually 2x) frequency components are available. Figure 2-4 shows an example of this clock generation method. Here, a 25 MHz oscillator is used with a clock generation circuit to produce the 25 MHz clock for the memory and I/O bus modules. This 25 MHz clock is locally stepped up to 50 MHz using this new type of clock chip providing a local, synchronous, 50 MHz clock which has a minimal, controlled skew referenced to the 25 MHz clock. There are several advantages to this scheme: first a lower frequency crystal is required; second, if multiple 50 MHz modules are required, such as a second CPU module for a multi-processing system, a central 50 MHz signal need not be routed and deskewed across a large physical distance. A drawback to this scheme is it's relatively new and there are a limited number of clock chip vendors. Also, as with other clock generation methods, it is important to verify the 50 MHz clock meets the stability specification of the components.

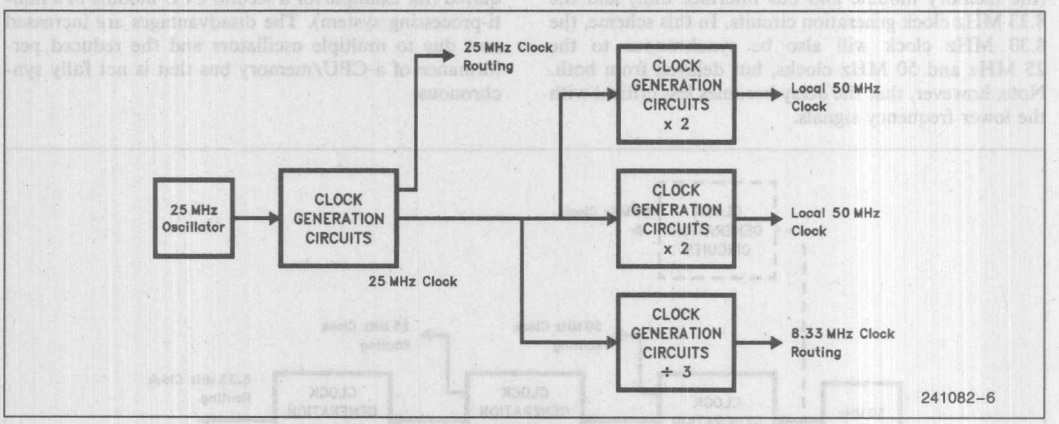


Figure 2-4. Local Generation of Synchronous Clocks

With high frequency systems, controlling component to component skew will be one of the greatest challenges of system clock designs. There are numerous sources of clock skew as outlined below:

1) Intrinsic Skew—Clock skew generated from the clock chip—
   A) Rising edge delay to falling edge delay skew on the same output (often called pin skew—t_{ps}).
   B) Skew between the delay from one output to another output on the same chip (often called output skew—t_{os}).
   C) Output to output skew between different chips.
2) Extrinsic skew—Clock skew generated from routing/loading—
   A) Skew generated by different rise or fall times.
   B) Skew from different tap points.
   C) Skew from flight time differences as a result of trace length or loading differences.
   D) Clock skew generated from poor signal waveform integrity.

Reference 4 covers the different components of skew generated by the inherent limitations within the chip design. These skews can only be changed by choosing different clock chips. The second class of skew, extrinsic skew, are under the control of the system designer.

Figure 2-5 illustrates some of these skew components. Signals A and B have the same loading, however, the resulting waveforms at the destination have a skew. This is due to the intrinsic buffer skew, t_{os}. Signal C has twice the capacitance of signal A and the extra delay and signal rise time result in an even greater skew between these two signals. Finally, signal D must pass through a transmission line and the resulting *flight time* delay causes an even greater skew between itself and signal A.

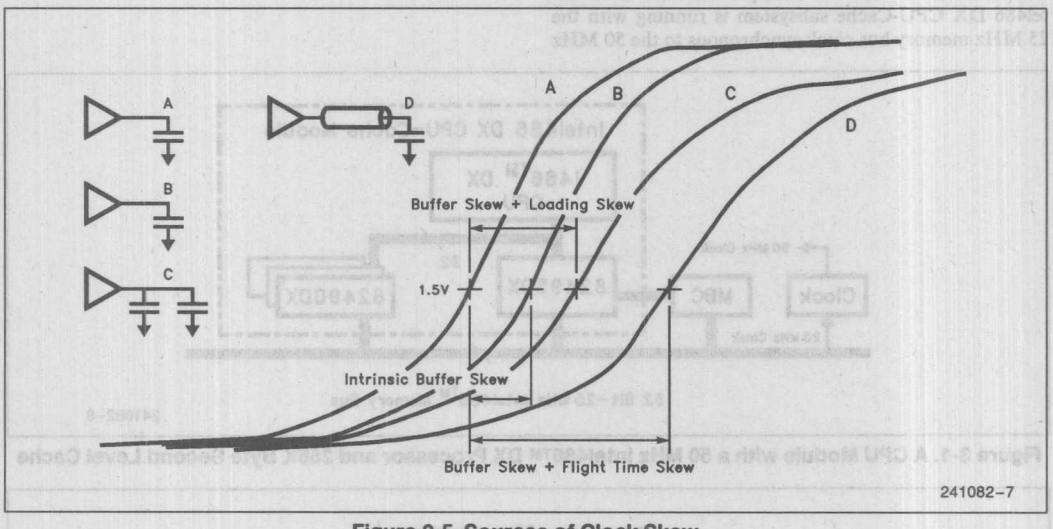


Figure 2-5. Sources of Clock Skew

3.0 DESIGNING A SYSTEM CLOCK

Designing a clock distribution requires the careful examination of a series of issues:

1) Choosing a clock driver.
2) Determining the layout and loading of the clock signal on the board.
3) Choosing the type of termination.
4) Simulating the expected performance of the clock signal on an electrical simulator (such as SPICE).

3.1 Design Example

This application note will work through each of these steps using the design of the CPU subsystem shown in Figure 3-1 below. Here, a 50 MHz Intel486 DX CPU-Cache module is interfaced to a 25 MHz memory bus with a memory bus controller and a clock generation and distribution chip. The example will assume the Intel486 DX CPU-Cache subsystem is running with the 25 MHz memory bus clock synchronous to the 50 MHz CPU core clock. The 82495DX and 82490DX support this directly (divided synchronous mode).

The 82495DX and 82490DX components provide a highly efficient 128K Byte, 256K Byte or 512K Byte second level cache. One 82495DX cache controller and 8 (or 9 for parity) 82490DXs implement a 256K Byte cache core, requiring only a memory bus controller to interface to a high speed system memory bus. The Intel486 DX CPU-Cache module is a multi-package module (MPM) which implements a 256K Byte cache with parity along with an Intel486 DX microprocessor. This board can interface to the rest of the system through a special 240-pin impedance matched connector.

The Intel486 DX CPU-Cache chip set, operating at 50 MHz, has the clock requirements shown below in Figure 3-2. Skew between the 82495DX and Intel486 CPU can not exceed 0.5 ns (measured at 1.5V), while skew between the 82490DX and either the 82495DX or Intel 486DX CPU may not exceed 1.0 ns.

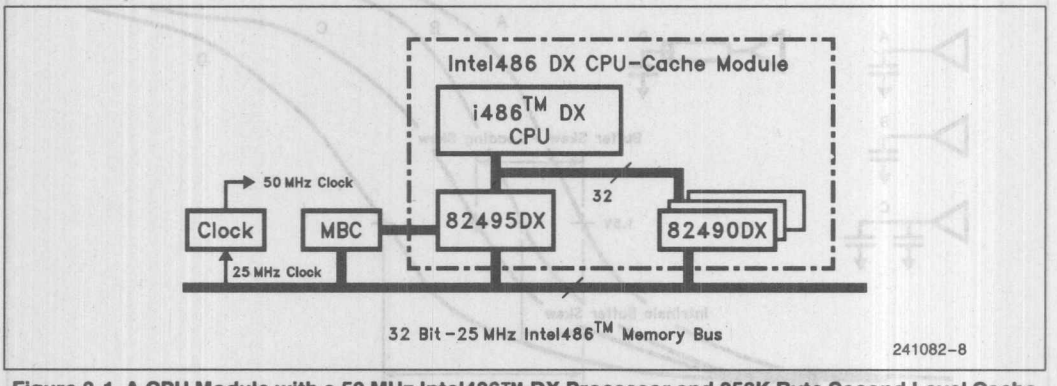


Figure 3-1. A CPU Module with a 50 MHz Intel486™ DX Processor and 256K Byte Second Level Cache

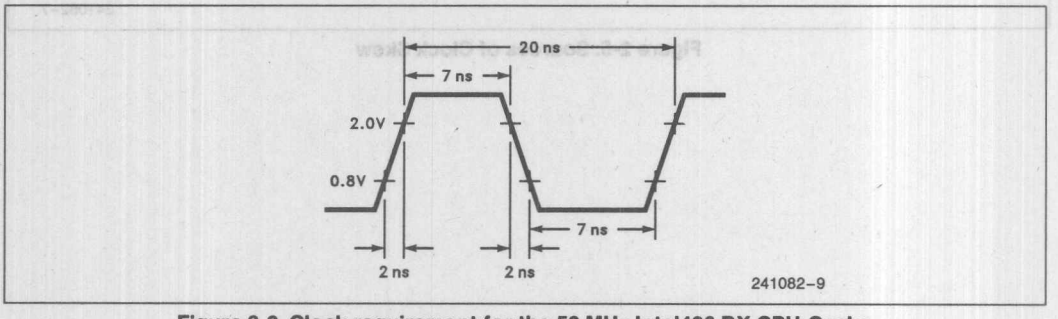


Figure 3-2. Clock requirement for the 50 MHz Intel486 DX CPU-Cache chip set and module. The 7 ns and 20 ns valid and period times are minimum values and the 2 ns rise and fall times are maximum values.

Figure 3-3 shows a conceptual model of the clock distribution to the Intel486 DX CPU-Cache subsystem. A number of elements are outlined: a clock driver chip; a series of clock traces on the motherboard which lead to the module's connector; the impedance matched connector itself; the traces on the module; and, finally the components themselves. For a first order investigation, the module traces shown actually reflect the branching and loading of the traces implemented on the module. For the module, the maximum clock skew is specified at the connector inputs.

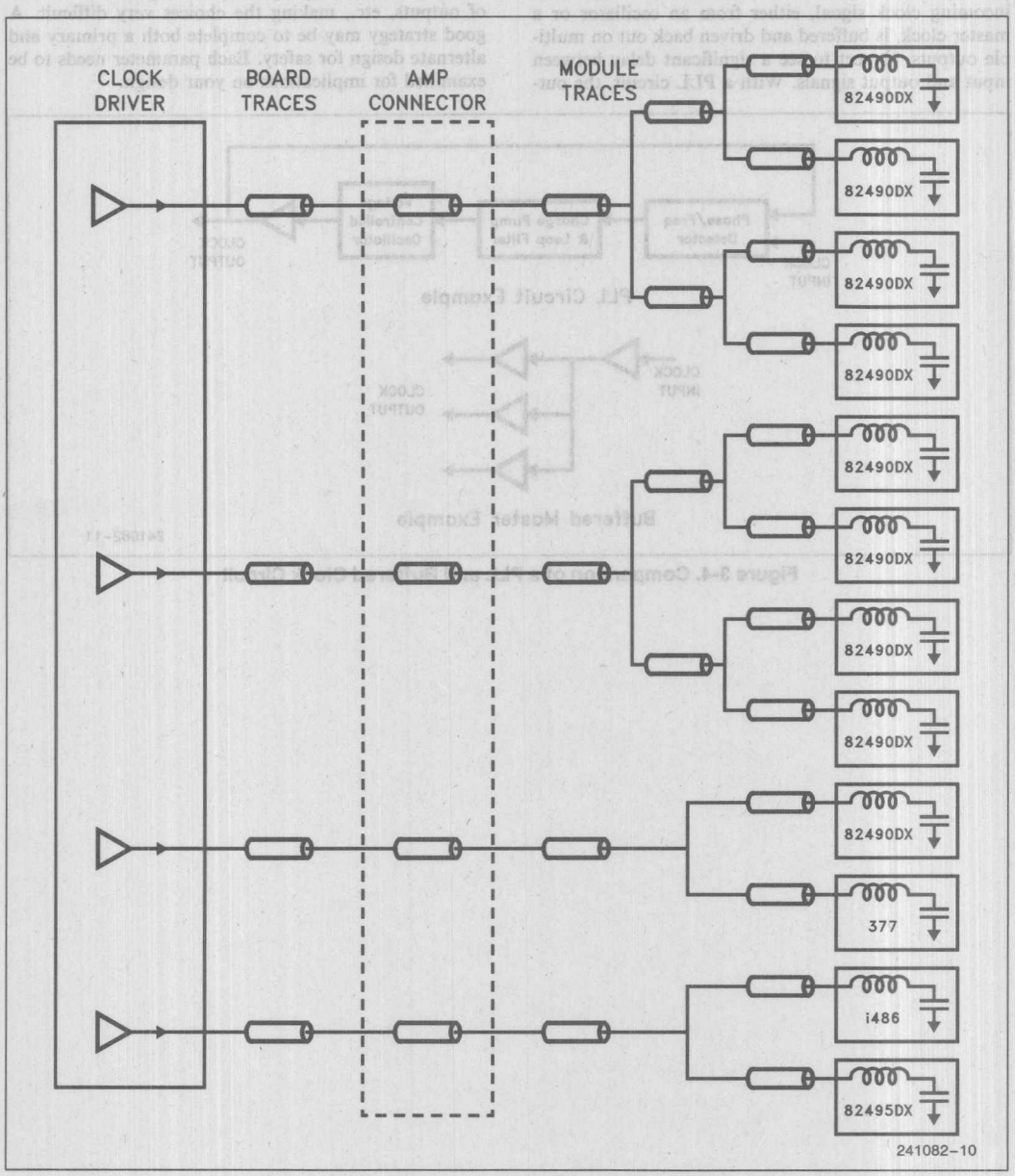


Figure 3-3. Model of the Clock Distribution to the Intel486 DX CPU-Cache Subsystem

3.2 Choosing a Clock Driver

There are two basic types of chips available: buffered master clocks and phase locked loop synchronized clocks. Figure 3-4 shows the differences between these two types of components. With the buffer version, an incoming clock signal, either from an oscillator or a master clock, is buffered and driven back out on multiple outputs. Expect to see a significant delay between input and output signals. With a PLL circuit, the output clock is synchronized to the input clock resulting in very little input to output delay.

Table 3-1 shows some of the driver chips which may be used for clock generation. Clearly, there is no standard in any parameter, type, package, voltage levels, number of outputs, etc., making the choices very difficult. A good strategy may be to complete both a primary and alternate design for safety. Each parameter needs to be examined for implications on your design.

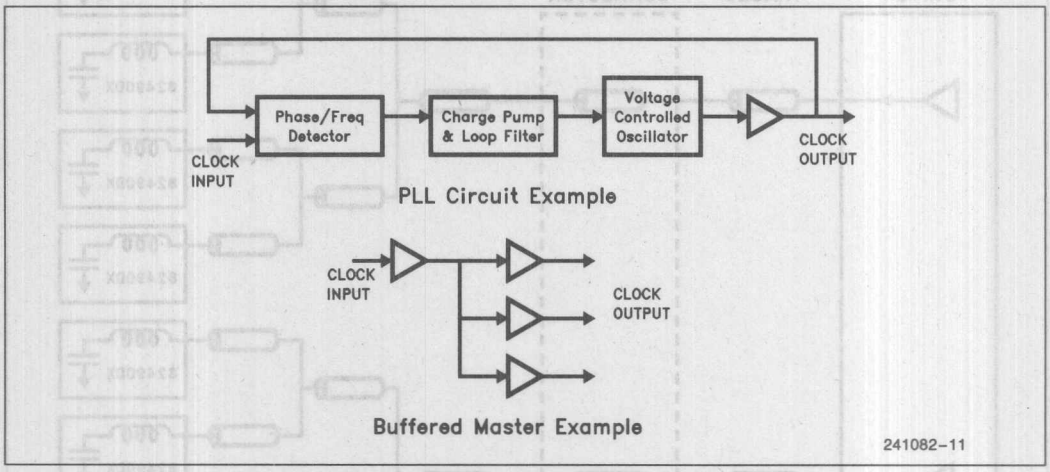


Figure 3-4. Comparison of a PLL and Buffered Clock Circuit

Table 3-1. Clock Driver Chip's Specifications

Manufacturer(1)	Device	Technology	Type	Package	Max Output	t_{os}	Rise/ Fall	Max Delay	Level In/Out	Clock Out @2
Texas Instruments	74ACT 11208	CMOS	Buffer	20-Pin DIP LCC/Plastic or Ceramic	NA(3)	1 ns	NA	NA	TTL/CMOS	4 @ 1x and 4 @ 1x
IDT	49FCT805A 49FCT806A	CMOS	Buffer	20-Pin DIP, SOIC, CERPACK, LCC	NA	0.7 ns	NA	5.8 ns	TTL/CMOS	5 @ 1x and 5 @ 1x
Silicon Connection	SC3501	BiCMOS	Buffer	52-Lead PGFP	80 MHz	0.5 ns(4)	2 ns/ 2 ns	NA	PECL or TTL/TTL	10 @ 1x 5 @ 1x or 0.5x 5 @ 0.5 or 0.25x
Motorola(5)	MC88915	NA	PLL	28-Pin PLCC	80 MHz	0.5 ns	2.5(6) @ 50 pF	0.5 ns	TTL/CMOS	5 @ 1x and 1 @ 2x and 1 @ 0.5x
Gazelle	GA 1110	GaAs	PLL	16-Pin DIP PLCC Due	50 MHz	0.5 ns	1.5	0.5 ns	TTL/TTL	6 @ 1x
Gazelle	GA 1210	GaAs	PLL	16-Pin DIP PLCC Due	100 MHz	0.5 ns	1.5	0.5 ns	TTL/TTL	2 @ 1x and 4 @ 2x
National Semi.	CGS74 CT2525	FACT	Buffer	14-Pin PDIP SOIC	70 MHz	0.7 ns	2	12.4 ns	TTL/TTL or CMOS	8 @ 1x

NOTES:
1. Check with manufacturers for the most accurate information.
2. Number or clock outputs specified as a fraction of the input clock frequency. For example 5 @ 0.5x indicates there are 5 copies available at ½ the input frequency.
3. NA indicates the data is not available on the data sheet.
4. 0.25 ns skew within a group of 5 outputs.
5. This device requires 1 external resistor (1.0 KΩ) and 2 external capacitors (0.01 μF and 5.0 nF).
6. Rise and fall times are 0.2 V_{CC} to 0.8 V_{CC}.

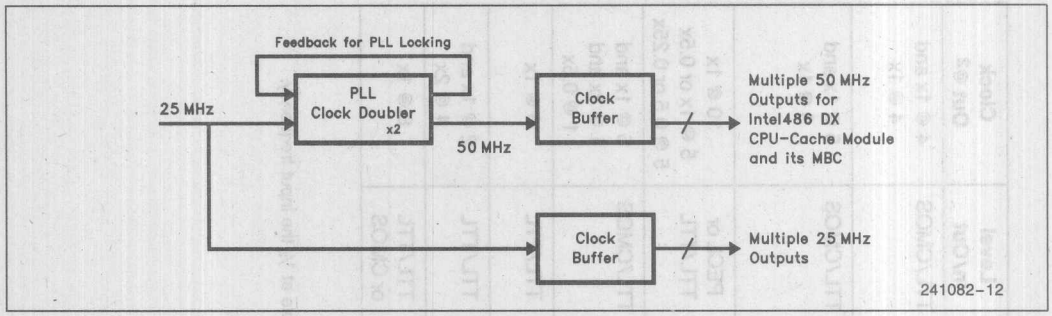


Figure 3-5. Design Example for Generation of Multiple Synchronized 2x Clocks

Packaging is important because of the board area available and the need (or inability) to use surface mount. Maximum skew and rise/fall times are important in meeting the Intel486 DX CPU-Cache chip set specification. Also, a skew of 0.5 or 0.7 ns uses most of the skew budget at the clock driver and leaves little margin for routing generated skew.

When considering the rise and fall time specification (which many manufacturers do not specify in their preliminary data sheets) it is important to understand the effect of additional loads. Only the Motorola 88915 specified the rise and fall times at a given capacitance.

Delay will be an important consideration for the synchronization required. The memory bus controller may be sensitive to skew between the memory bus clock and the CPU core clock.

Finally, the number of outputs available, their drive capability and the relative frequency is critical. Insufficient numbers of copies will require multiple chips, introducing synchronization and chip to chip skew issues. It is critical to determine the maximum load any single output can drive while maintaining the clock requirements shown in Figure 3-2.

A technique for driving additional load without suffering the skew resulting from using another clock driver output is to connect together multiple clock outputs. Note, however, this technique may not work with certain clock driver chips. Consult the data sheet or the clock manufacturer before using this technique.

For the design example an input 25 MHz clock must be stepped up to 50 MHz and distributed to the module and the memory bus controller. The Intel486 DX CPU-Cache module requires four separate clock inputs. The memory bus controller requires at least one input and preferably more. One possible design is shown in Figure 3-5. Here a PLL clock chip is used as a clock doubler to generate a synchronous 50 MHz clock. This 50 MHz signal is then buffered with another clock driver chip to produce multiple 50 MHz clocks.

3.3 Clock Signal Layout

The most critical aspects of signal routing are maintaining equal route lengths and loads for equal flight times. Figure 3-6 shows several methods of routing clock signals. These include distributing the loads along clock route at various tap points, building a star configuration and designing a series of branches. The tapped layout, while often minimizing routing, suffers from unequal flight times of the clock to the various tap points. Thus, it would be difficult to control skew between the first and last loads. With the star or branched layouts, the flight distance of each leg can be carefully matched.

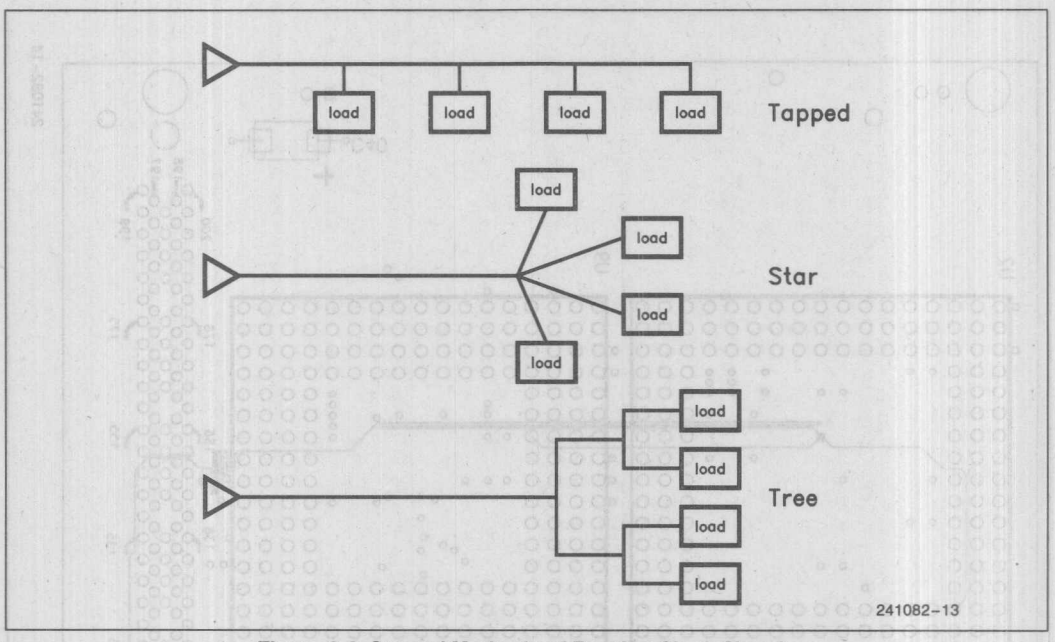


Figure 3-6. Several Methods of Routing Clock Signals

Our analysis and experiments show that right angles in traces, vias and other similar discontinuities produce rather minimal effects. They should be avoided if possible, but are not disastrous. At frequencies above 50 MHz, however, these layout parameters will become much more crucial.

Figure 3-7 shows the routing of the clocks on the Intel486 DX CPU-Cache module. The layout is carefully designed to eliminate any flight time skew. This leaves the routing up to the connector and the routing to the MBC as the critical components remaining. The Intel486 DX CPU-Cache module clocks are designed such that equal traces to the connector will result in near zero flight time skews. Thus, routing to the MBC will be the most challenging aspect. Applying the same ideas discussed here will result in minimal flight time skews.

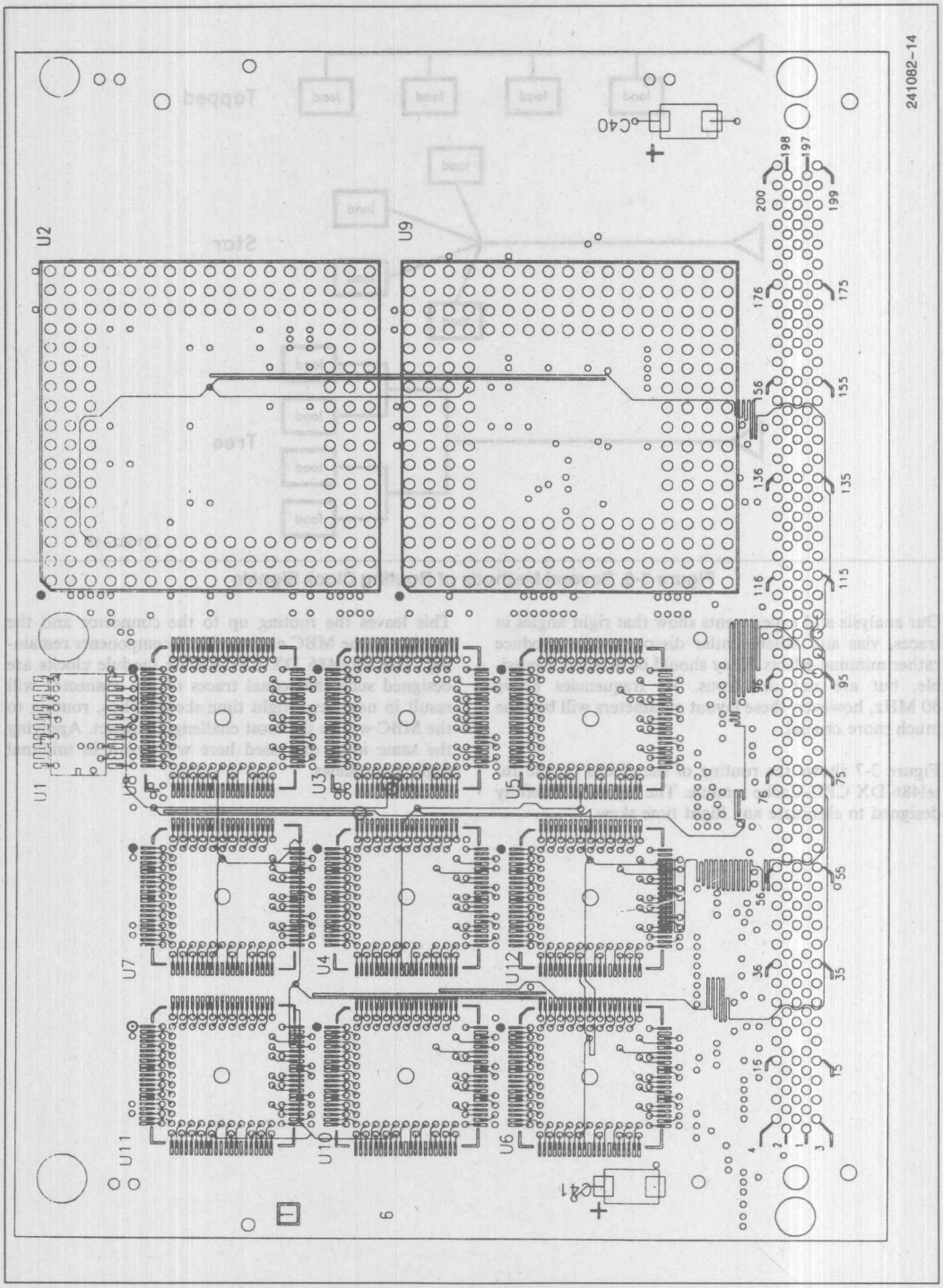


Figure 3-7. Routing of the Clocks on the Intel486 DX CPU-Cache Module

3.4 Termination

Termination of the clock signals is a critical design parameter. The module's traces and connector are well matched at 50Ω. If the impedance of the board traces from the clock chip to the connector are controlled to 50Ω, the remaining variable will be the output impedance of the clock driver chip. Unfortunately, determining this may not be a trivial procedure. The first step would be to contact the clock manufacturer and determine if they supply the clock driver's output impedances or I/O models of the driver. If not, Appendix A outlines three methods of estimating the driver's output impedance.

There are two basic problems with matching a driver's output impedance. First, the driver may have a different pull-up to pull-down output impedance. This drastically complicates termination. Second, output impedances are typically not specified on the clock chip data sheets. Thus they are not guaranteed by the manufacturer's testing programs.

If the driver's output impedance matches the impedance of the clock traces, no termination is required. However, this is not likely to be the case. If termination is required, series termination will probably be the best choice. Reference 5 outlines the various approaches and concludes that series termination is the best design choice for star configuarations.

Ideally, series termination into a 50Ω transmission line with a star configuration should be 50Ω minus the driver's impedance. In practice a combined impedance of 35Ω–50Ω may yield acceptable results. The driver itself can range from 7Ω–35Ω. After determining the output impedance of the clock chip being used, the appropriate series termination can be chosen. For example, if the clock chip being used has a 10Ω output impedance, then a 25Ω–40Ω series resistor may be needed. If the clock chip's output impedance is 35Ω, then series termination may not be necessary. Note that if your clock routing is based on a taped configuration, then series termination may not be adequate.

3.5 Simulation

The initial design choices will need simulation with a circuit simulator such as SPICE for final verification. There are a number of issues which need to be addressed:

1. Methods to model the transmission lines.
2. Generation of a netlist from layout database.
3. Modeling the clock driver.
4. Modeling the connector if the module is being used.
5. Input models of the Intel486, 82495DX and 82490DX components, clock inputs or the input model of the module's clock pins.

The transmission line can be modeled in one of two ways, depending on the simulator chosen. First, if the simulator has a transmission line model built into it, this will greatly simplify the task. Care should be taken to verify the accuracy of this model by contacting the simulator's vendor. If the transmission line model is available, the line segment can be simply modeled as a separate component with parameters of length and characteristic impedance as show below

tr1 first__node second__node 1.2 in. 50 ohm

where first__node and second__node are the netlist names of the segments connections, 1.2 in is the length of the segment in inches and 50Ω is the characteristic impedance. If the simulator does not have a built in transmission line model, discrete inductors and capacitors can be used. The values of the inductors and capacitors will vary with the board's characteristics and should be supplied by the board manufacturer. Intel's analyses have shown that at least one inductor and capacitor pair per ⅓ inch of PC trace yields sufficient accuracy. The example below shows a PC trace modeled by a capacitor and inductor.

l1 first__node second__node 25 nh
C1 second__node vss 1.0 pF

The netlist can be automatically generated by most PC board design tools. In addition, most tools will supply a switch which allows PC traces to be broken into transmission line or L-C segments. If designing with the Intel486 DX CPU-Cache module, reference 6 will give a detailed model of the module's clock traces.

When attempting to model the clock driver, first contact the manufacturer for I/O models. If these are not available, a simple model must be constructed. First, use the method outlined in Appendix A to estimate the output impedance of the driver. This can be used in the simple model shown in Figure 3-8. The last element of this simple model is the voltage source. The rise time of this element is equivalent to the rise time measured on an unloaded clock driver chip. Also, note that for many drivers, the output resistance and driver dv/dt may differ between the rising and falling transitions.

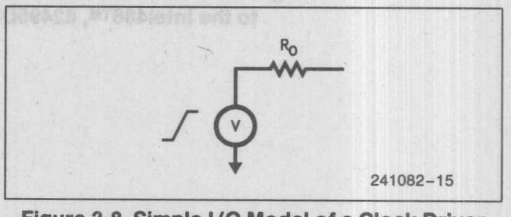


Figure 3-8. Simple I/O Model of a Clock Driver

If the Intel486 DX CPU-Cache module is being used, the connector must also be accurately modeled. AMP provides a SPICE model for all module customers (Ref-

erence 7). This model emulates the impedance matching characteristics of the connector and faithfully reproduces any mutual inductance effects.

Finally, the input pins of the Intel486 CPU, 82495DX and 82490DX components must be modeled. The Intel reference 8 accurately specify models for these pins.

With all of the pieces now modeled, the task turns to simulating the network and evaluating waveforms for skew and quality. In addition to meeting the clock specifications shown in Figure 3-2, ringback and settling time must be evaluated for spurious levels which would violate the clock input limits.

An important consideration when modeling the input circuits of typical Intel components is the inclusion of diodes to the supplies. Most CMOS components including the Intel486 microprocessor, 82495DX and 82490DX contain these diodes to prevent damage to the input buffers due to electrostatic discharge. When first simulating the clock circuits it is advisable to not include these input diodes. The reason is their clamping action could hide damaging amounts of energy being driven into the input pins of the components. Final simulations could include these diodes in order to more closely model the actual waveform shapes.

Figure 3-9 shows the result of a simulation of the clock signals delivered to the Intel486, 82495DX and 82490DX components on an Intel486 DX CPU-Cache module. The simulation is for a hypothetical clock driver modeled as shown in Figure 3-8 and unloaded waveform as shown for UCLOCKE in Figure 3-9. This simulation shows the clock signals meeting all of the input clock specifications outlined in Figure 3-2.

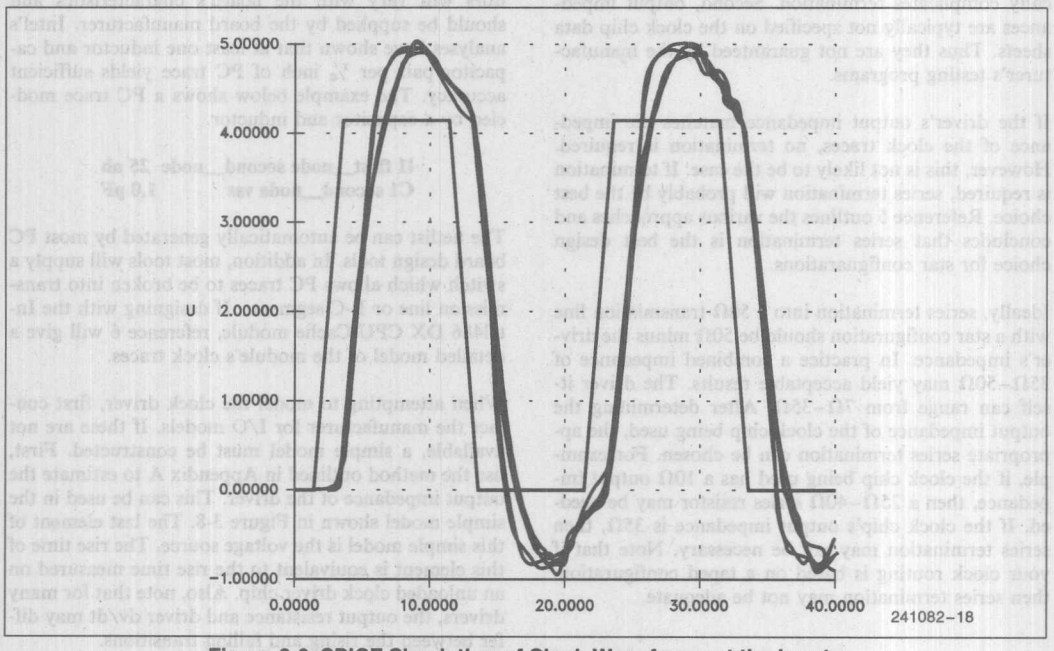


Figure 3-9. SPICE Simulation of Clock Waveforms at the Inputs to the Intel486™, 82495DX and 82490DX Components

4.0 SUMMARY

This application note has discussed techniques for the design of clocks in high speed, 50 MHz Intel486 systems. At these frequencies, edge rates and trace lengths cause routes to behave as transmission lines with their associated characteristics. For the system designer, clock routing issues become important enough to affect the overall system design.

A second level cache such as implemented by the 82495DX and 82490DX components provides the potential for a *frequency isolation* function by decoupling the memory bus speed and the CPU bus speed. Thus, hierarchies of buses can be produced, isolating the difficult 50 MHz signals.

A number of clock generation and distribution techniques were discussed including a traditional centrally generated master clock, local asynchronous clocks and the new technique of locally regenerated synchronous clocks. This latter techniques is made possible by a new class of clock chip which have internal phase locked loops.

The sources of clock skew were outlined and broken into two major categories: intrinsic skew, which are attributable to the clock generation chip; and extrinsic skew, which is caused by the routing and loading of the clock lines themselves.

To address intrinsic skew, various clock drivers were compared and contrasted. There is no clear choice, as different clock chips fit various system configurations.

To address extrinsic skew, clock signal layout and termination were discussed. Either a star or tree clock layout with balanced loads is recommended. Series source termination is also recommended. The value of the termination resistor will depend on the characteristic impedance of the PCB traces and the output impedance of the clock driver.

Finally, electrical simulation with a tool such as SPICE becomes essential at these speeds. Methods to model transmission lines, generate a netlist, model the clock driver and module connector and inputs to the Intel486 DX, 82495DX and 82490DX components were discussed.

5.0 REFERENCES

1. *33 MHz Intel386™ System Design Considerations*, Intel Application Note. AP-442, Order Number: 240725-001
2. Intel486 DX CPU-Cache Chip Set Datasheet, Intel Corporation, Order Number 241084.
3. Intel486 DX CPU-Cache Module Datasheet, Intel Corporation, Order Number 241083.
4. Clock Distribution Design Success with Advanced Logic, R. Morgan, Wescon 1990 Digest, pp. 35–40.
5. Advanced High Speed CMOS Logic in a Transmission Line Environment, R. Funk and E. Wittmann, Wescon 1990 Digest, pp. 29–34.
6. Intel486 DX CPU-Cache Module Hardware Reference Manual, Intel Corp., Order Number 241091.
7. SPICE model of AMP connector. AMP Corporation. (717) 986-7468.
8. Intel486 DX CPU-Cache Chip Set Hardware Reference Manual, Intel Corp., Order Number 241172.

2

General References on Transmission Line Effects

Advanced CMOS Logic Designer's Handbook, Texas Instruments Inc., 1988.

Blood W., *MECL System Design Handbook*, Motorola Corp., 1983.

Keeler R., "High Speed Digital Printed Circuit Boards," *Electronic Packaging & Production*, pp. 140-145, Jan. 1986.

Tomlinson J., "Avoid the Pitfalls of High Speed Logic Design," *Electronic Design*, pp. 75–84, Nov. 9, 1989.

Pace C., "Terminate Bus Lines To Avoid Overshoot and Ringing," *EDN*, pp. 227–234, Sept. 17, 1987.

Royle D., "Rules Tell Whether Interconnections Act Like Transmission Lines," *EDN*, pp. 131–136, June 23, 1988.

Royle D., "Correct Signal Faults By Implementing Line-Analysis Theory," *EDN*, pp. 143–148, June 23, 1988.

Winchester E., "Guidelines Help You Design High-Speed PC Boards," *EDN*, pp. 221–226, Nov. 28, 1985.

Yeargan J. R., Day R. L., and Nguyen T., "Effects of Printed Circuit Board Transmission Lines and Loading on Gate Performance," *IEEE Transactions on Industrial Electronics*, Vol. IE-34, no. 3, pp. 399–405, Aug. 1987.

Clock Chip Application Notes

1. SC3501—20 Copy Clock Driver. Silicon Connections Corporation.
2. Why 20 Clock Copies? Silicon Connections Corporation.
3. Clock Distribution Simplified with IDT Guaranteed Skew Clock Drivers, Michel Conrad. Integrated Device Technology. AN-82.
4. Low Skew Clock Drivers and Their System Design Considerations, Motorola Semiconductor Application note AN1091.
5. Clock Distribution Techniques Using the GA 1110. Gazelle Microcircuits, Inc. Application Note 1.

Clock Chip Manufacturers

1. Texas Instruments
   Dallas, TX
   Contact: Rick Curtis
   Phone: (903) 868-5628
2. Integrated Device Technology, Inc. (IDT).
   3236 Scott Blvd.
   PO Box 58015
   Santa Clara CA 95052-8015
   Phone: (408) 727-6116
   Contact: Kiran Kapshikar
3. Silicon Connections Corporation.
   16868 Via Del Campo Ct.
   San Diego, CA 92127
   Phone: (619) 674-1050
   Contact: Bill Smith
4. Motorola Inc.
   Contact: Willard Tu (602) 962-2657
5. Gazelle Microcircuits, Inc.
   2300 Owen Street
   Santa Clara, CA 95054
   Phone: (408) 982-0900
   Contact: K. Annamalai (x147)
6. National Semiconductor
   Santa Clara, CA
   Contact: Tony Ochoa
   Phone: (408) 721-6804

APPENDIX A
METHODS TO MEASURE THE OUTPUT IMPEDANCE OF A CLOCK DRIVER

Method 1) I-V Characteristics.

In this method, the driver's impedance is determined by measuring the output I-V. Figure A-1 illustrates this method for determining the pull-up impedance. The 50Ω load line is drawn and the intercept with the output driver characteristic is also the intercept of a second load line drawn through the initial voltage of the driver.

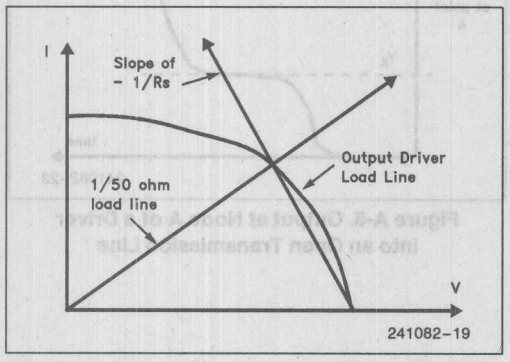


Figure A-1. Extraction of Output Driver Pull-Up Impedance with the I-V Method

Method 2) Derating Curve.

The second method of estimating the clock driver's output impedance is diagramed in Figures A-2 and A-3. The 10% to 90% rise time is measured for a range of load capacitances. Plotting the resulting data will yield a line whose slope is proportional to the driver's output impedance. A derivation of the slope (and the constant k) is shown below:

$$t_r = t_o + kRC$$

where t_o = the unloaded rise time (the y intercept in Figure A-3)

$$\frac{dt_r}{dC} = kR$$

let $\tau = RC$

$$V(t) = V_f\left(1 - e^{-t/\tau}\right)$$

$$0.1 = \left(1 - e^{-t_{10\%}/\tau}\right) \quad \text{then } t_{10\%} = 0.105\tau$$

$$0.9 = \left(1 - e^{-t_{90\%}/\tau}\right) \quad \text{then } t_{90\%} = 2.303\tau$$

$$t_{90\%} - t_{10\%} = (2.303\tau - 0.105\tau) = 2.198\ RC$$

$$k = 2.198$$

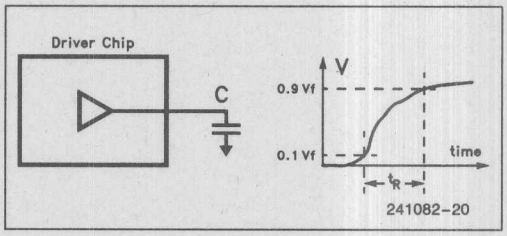


Figure A-2. Test Circuit and Response for the Derating Curve Method of Extracting the Output Driver's Impedance

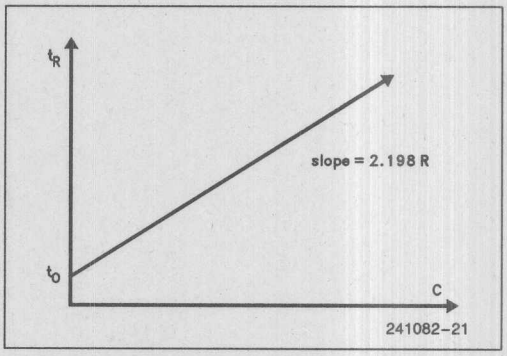


Figure A-3. The derating curve of an output driver. The slope is proportional to R.

2

Method 3) Open Transmission Line.

Another method uses the clock chip driving an open transmission line of known impedance as shown in Figure A-4. The voltage will initially rise to the value V_x at point A as shown in Figure A-5. This is the result of the voltage divider across the driver and transmission line

$$\frac{V_x}{V_F} = \frac{Z_L}{R_s + Z_L}$$

Once the wave reaches the endpoint of the transmission line it reflects back to the source. The wave will continue back and forth several times until the voltage at node A eventually rises to its final value, V_F. With V_x,V_F and Z_L known R_s can be determined.

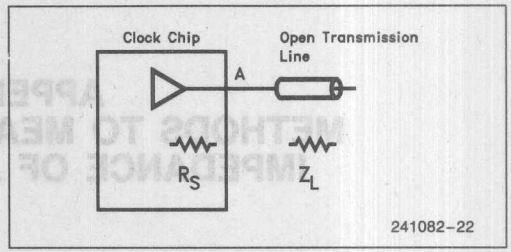


Figure A-4. Test Circuit Used to Extract the Output Impedance with the Open Transmission Line Method

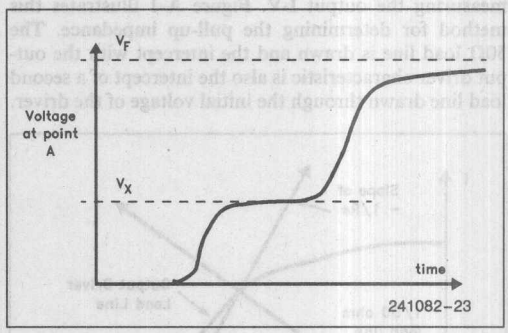


Figure A-5. Output at Node A of a Driver into an Open Transmission Line

intel®

AP-458

APPLICATION NOTE

Designing a Memory Bus Controller for a 50 MHz Intel486™ DX CPU-Cache System

2

TAUFIK MA
ISIC SILAS
TECHNICAL MARKETING

October 1991

Order Number: 241166-002

Designing a Memory Bus Controller for a 50 MHz Intel486™ DX CPU-Cache System

CONTENTS

PAGE

1.0 OVERVIEW OF APPLICATIONS NOTE 2-955

2.0 INTRODUCTION TO A SECOND LEVEL CACHE SOLUTION 2-955

2.1 Main Components of the Cache Subsystem 2-955

3.0 MEMORY BUS CONTROLLER DESIGN ALTERNATIVES 2-956

3.1 MBC Architectural Model 2-956

3.2 MBC implementation Model 2-957

3.2.1 Cycle Control Block 2-957

3.2.2 Data Path Control Block 2-957

3.2.3 Snoop Control Block 2-958

3.2.4 Test Logic 2-958

4.0 DECISION PROCESS 2-958

4.1 Range of Memory Bus Implementations 2-959

4.2 Choosing a Memory Bus 2-959

4.2.1 Cache Protocol 2-960

4.2.2 Memory Bus Width and Speed 2-962

4.2.3 Memory Bus Arbitration 2-962

4.2.4 Memory Bus Protocol 2-962

4.2.5 Snooping Mode 2-962

4.3 Choosing Cache Parameters 2-962

4.4 Other Assumptions 2-963

5.0 DESIGN PROCESS 2-963

5.1 Clock Generation 2-963

5.2 Reset and Initialization 2-964

5.3 Logic Design 2-965

5.3.1 82495DX Cycles versus Intel486™ DX Cycles 2-966

5.3.2 Requests for the Memory Bus 2-968

5.3.3 Starting a Cycle 2-968

5.3.4 Completing the Cycle 2-969

5.3.4.1 Controlling Cacheability 2-969

5.3.4.2 Supporting Multiple Transfer Requirements 2-969

5.3.4.3 Controlling the Data Flow 2-971

5.3.5 Snoopee Cycles 2-973

5.3.5.1 Snoopee Response in Write-Through Mode 2-974

5.3.5.2 Snoopee Response in Write-Back Mode 2-974

5.3.6 Snoopee Write-Back Cycles 2-974

5.3.7 Snooper Requirements 2-975

5.3.8 Special Cycles 2-975

CONTENTS

PAGE

5.3.8.1 I/O cycles 2-975
5.3.8.2 FLUSH/SYNC cycles 2-975
5.3.8.3 LOCK cycles 2-975
5.3.8.4 PLOCK cycles 2-976
5.3.9 Warm Reset 2-976
5.3.10 MHOLD/MHLDA Arbitration 2-977

6.0 TIMING DIAGRAMS 2-979

7.0 SCHEMATICS AND PLD CODES 2-983

8.0 APPENDIX 2-1002
8.1 Configuration during Reset 2-1002
8.2 Worse Case Timing Analysis 2-1002
8.2.1 Clock Skews 2-1002
8.2.2 Module Input Signals 2-1002
8.2.3 Module Output Signals 2-1003

9.0 SUGGESTED OTHER READING 2-1003

2

FIGURES

Figure 2-1 Block Diagram of an Intel486™ DX CPU Solution With a Second Level Cache 2-955
Figure 3-1 The Architectural Model for the Memory Bus Controller 2-956
Figure 3-2 Implementation Model of a Memory Bus Controller 2-957
Figure 4-1 This Design Example Builds an Intel486™ DX 'Super' CPU Module 2-960
Figure 4-2 MESI State Transitions 2-961
Figure 4-3 MESI Simplified to Support a Write-Thru Policy 2-961
Figure 4-4 The Choice of Memory Bus Width and Cache Size are Coupled 2-963
Figure 5-1 The MBC Uses a Divided-Synchronous Design 2-964
Figure 5-2 The 82495DX Cache Controller Progresses Through a Cycle 2-966
Figure 5-3 Block Diagram of the Intel486™ DX Simple MBC 2-967
Figure 5-4 The XMBReq State Machine for the MBC 2-968
Figure 5-5 The XMADS State Machine for the MBC 2-968
Figure 5-6 KWEND and SWEND are Generated by the XWND State Machine on the MCLK Following BGT 2-970
Figure 5-7 The XMBLAST State Machine Tracks Requests for Multiple Dwords 2-971
Figure 5-8 The XMADS State Machine Extended to Generate Multiple Address Strobes ... 2-972
Figure 5-9 Generating BRDY#S for the CPU-Cache (XBRDY & XDISBRDY) 2-973
Figure 5-10 Generating Ready's for the Cache Subsystem (XCRDY) 2-973
Figure 5-11 A Snoop Request Can Abort a Cycle Request (XMBReq) 2-975
Figure 5-12 The MBC Implements the LOCK and PLOCK Protocol (XMLOCK & XMPLOCK) 2-976
Figure 5-13 Generating a Warm Reset (XWRMRST) 2-977
Figure 5-14 The Design Example Using MHOLD/MHLDA for Bus Arbitration (XMHLDA) 2-977
Figure 5-15 The XMADS State Machine Modified to Account for Locked Cycles 2-978

CONTENTS

PAGE

FIGURES

Figure 6-1 Non-Cacheable Write Cycle 2-979
Figure 6-2 Non-Cacheable Read Cycle 2-980
Figure 6-3 Line Fill 2-981
Figure 6-4 Write Miss followed by Allocation Cycle 2-981
Figure 6-5 Ordinary Bus Exchange 2-982
Figure 6-6 Bus Exchange between Two Atomic Operations 2-982

TABLES

Table 3-1 Comparing Clocked and Strobed Memory Bus Implementations 2-958
Table 4-1 A Wide Range of Memory Bus Solutions are Possible 2-959
Table 5-1 Configuration Options 2-965
Table 5-2 MBC Cycle Translations 2-967
Table 7-1 State Machine Partitioning 2-983
Table 8-1 Configurations Options Set During Reset 2-1002

1.0 OVERVIEW OF APPLICATIONS NOTE

The design flexibility of Intel's high-performance second-level cache solution (82495DX/82490DX) allows a wide spectrum of implementations to match a diversity of system requirements. These implementations range from a simple write-through design suitable for single processor systems through to a comprehensive write-back design optimized for multiple processors. This Applications Note focuses upon the design process required to build the memory bus controller (MBC) that interfaces the second-level cache to the system's memory bus.

The example chosen as a vehicle to demonstrate this process is an MBC for a uni-processor Intel486 DX system. A design is presented that combines the 50MHz Intel486 DX CPU and 256KB-second level cache module with an MBC that interfaces to a 25MHz Intel486 DX-like memory bus. The resulting design can plug into the Intel486 DX socket of an existing system. Schematics and EPLD codes are included.

2.0 INTRODUCTION TO A SECOND LEVEL CACHE SOLUTION

Figure 2-1 shows a block diagram of an Intel486 DX microprocessor tightly coupled to a second level cache utilizing the 82495DX cache controller and several 82490DX dual-ported intelligent cache SRAMs. This cache subsystem was designed as a set of components and they interface gluelessly to the Intel486 DX microprocessor to provide zero wait state operation at 50MHz. They take full advantage of on-chip silicon flexibility to provide a degree of capability and performance otherwise unachievable with discrete implementations. The interface to the memory bus was designed with flexibility in mind and features may be selectively implemented via the MBC design to provide a wide range of price/performance implementations.

2.1 Main Components of the Cache Subsystem

The 82495DX cache controller forms the heart of the cache subsystem. The 82495DX includes the tag array with line state information to provide hit or miss decisions. It handles the CPU bus requests completely and coordinates with the memory bus controller when an access needs the memory bus; CPU accesses which can be serviced locally by the cache subsystem are filtered out from the memory bus traffic. The 82495DX monitors the memory bus and performs snoop operations when other bus masters are using the memory bus.

The 82490DX dual-ported intelligent cache SRAMs implement the data storage and data paths. They include latches, multiplexors and other logic which allow operation in lock-step with the 82495DX to efficiently serve both hit and miss accesses. They support zero wait state hit accesses and permit concurrent CPU and memory bus operations.

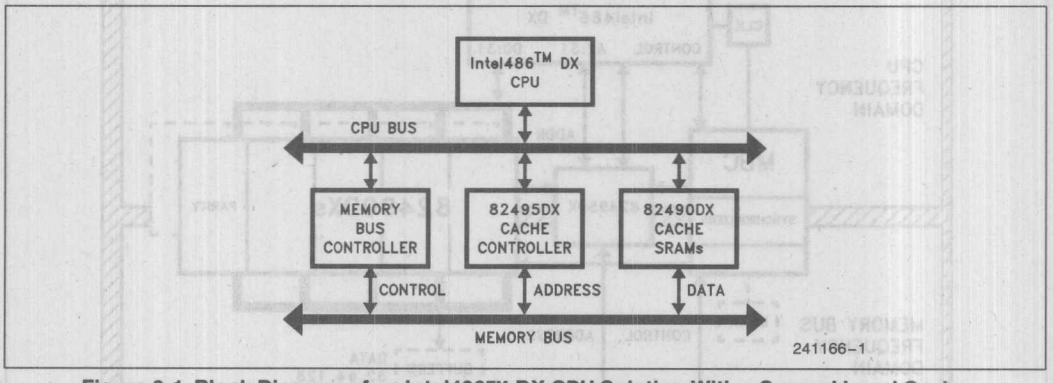


Figure 2-1. Block Diagram of an Intel486™ DX CPU Solution With a Second Level Cache

The Memory Bus Controller (MBC) is a user-definable array of logic that interfaces the CPU-Cache subsystem to a user-definable memory bus. The MBC adapts the CPU-Cache subsystem's generic protocol to a specific memory bus protocol. During the definition of this solution it was decided NOT to include a specific interface to a specific memory bus into the component implementation since such integration suffers from inflexibility and bandwidth limitations. The characteristics of the memory bus is pivotal to the performance and cost targets of a system solution, so if the memory bus were "hardwired" into the cache controller, it might be too costly for small systems and too slow for larger systems. With the memory bus interface implemented separately, it can be a complex ASIC for a high-bandwidth multi-processor system, or a few EPLDs for a basic implementation. Moreover, this key design point may be made into a product differentiator in the marketplace. The focus of this Applications Note is a step-by-step design of a 'typical' MBC - this Note is the first in a series of MBC implementation descriptions and covers the process of designing a Memory Bus Controller.

3.0 MEMORY BUS CONTROLLER DESIGN ALTERNATIVES

This section focuses upon the architectural model for an MBC and introduces some of the design flexibility of Intel's solution. The implementation alternatives are discussed and a framework for decision-making is built.

3.1 MBC Architectural Model

The architectural model for the MBC is shown in Figure 3-1.

Microprocessors are quickly achieving operating frequencies which can be very difficult for a memory subsystem to meet. A key part of the MBC model therefore includes the ability for the memory bus to operate asynchronously or at a different frequency from the CPU's clock frequency. This decoupling of the memory bus frequency from the CPU frequency allows a faster microprocessor to be included with no impact upon the memory bus design; the MBC must, of course, comprehend this higher CPU frequency.

Most of the MBC operates at the CPU's clock frequency, CLK. While it is possible to design the memory bus to operate at the same frequency as the 50MHz CPU clock, such a fast design is difficult to implement. The MBC architecture allows these high frequencies to be contained within the CPU-Cache-MBC core. The memory bus can operate in clocked or strobed modes.

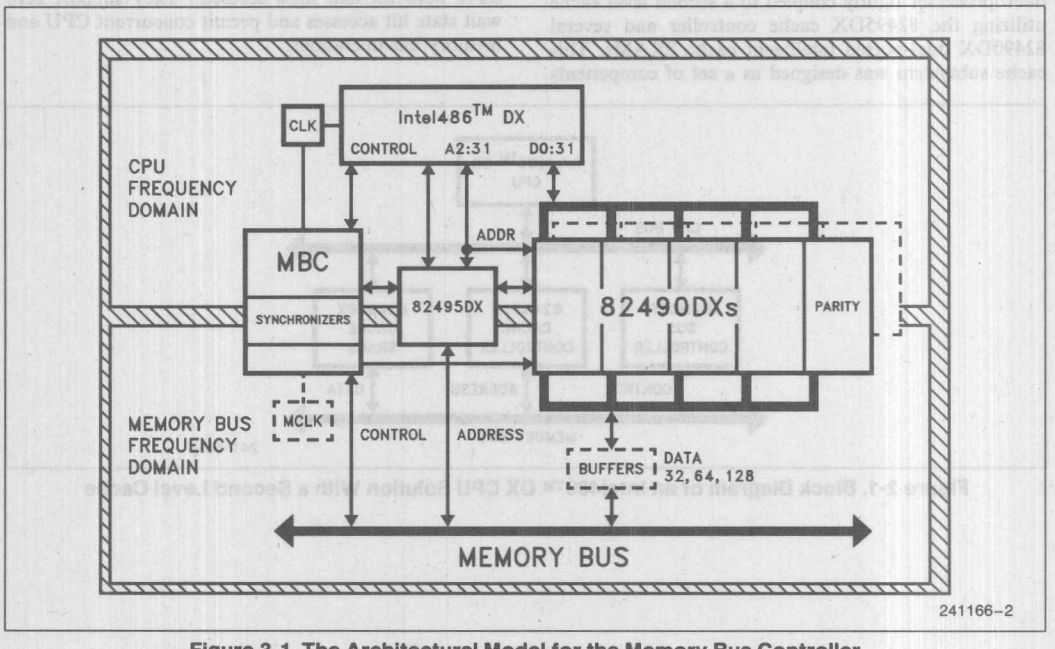


Figure 3-1. The Architectural Model for the Memory Bus Controller

In clocked mode the memory bus clock, MCLK, may be synchronous or asynchronous to CLK. The MBC defines synchronizers for the few signals which cross the two timing domains - these are not required in a synchronous design where MCLK is equal to or is a sub-multiple of CLK.

3.2 MBC implementation Model

An MBC is implemented in a few basic blocks as shown in Figure 3-2. The 82495DX cache controller requests control of the memory bus by signalling the MBC. The MBC is responsible for arbitrating and granting the bus to the 82495DX. Once granted, the MBC is responsible for executing the requested cycle, getting a snoop result from the other caches, and ending the cycle. The 82495DX supports different modes of snooping, different modes of memory bus operation, and various special cycles. The MBC design dictates which of these features are used, and exactly how they are used.

An MBC consists of a few basic blocks: a cycle control block (with synchronizers if necessary), a data path control block, a snoop logic block and a test block. The cycle control block must interface to some arbitration logic in order to take control of the memory bus and the snoop block must be able to communicate with the other caches when snooping is necessary.

3.2.1 CYCLE CONTROL BLOCK

Cycle control logic is responsible for initiating and terminating a memory bus cycle. Cycle control logic also provides cycle attributes such as cacheability, whether it is allocatable, pipelining, and all aspects of the progress of the current cycle.

The MBC supports all types of memory bus interfaces; the memory bus interface can be clocked or strobed. Table 3-1 details the engineering trade-offs for these types of memory buses.

Since cycle control logic interfaces memory bus signals to the 82495DX, and since the memory bus is not necessarily synchronous to the 82495DX CLK, it must also provide proper synchronization. Careful design of this synchronization logic can minimize or eliminate synchronization penalties.

3.2.2 DATA PATH CONTROL BLOCK

The data path control logic determines how data is written to, or read from, the 82490DX's. It handles the actual transferring of data to/from the memory data bus, including the CPU burst order and the control of data during allocation cycles. In systems with memory buses that are wider than the CPU bus, the data path control logic controls the different data requirements of the CPU and the 82490DXs.

The memory data bus can be 32-bits (the same as the Intel486 DX CPU), 64-bits or 128-bits wide; this will require four, eight or sixteen 82490DXs to implement the cache memory. This data cache memory may be parity protected; an 82490DX may be configured as a parity device and one can be used for up to each eight data cache devices.

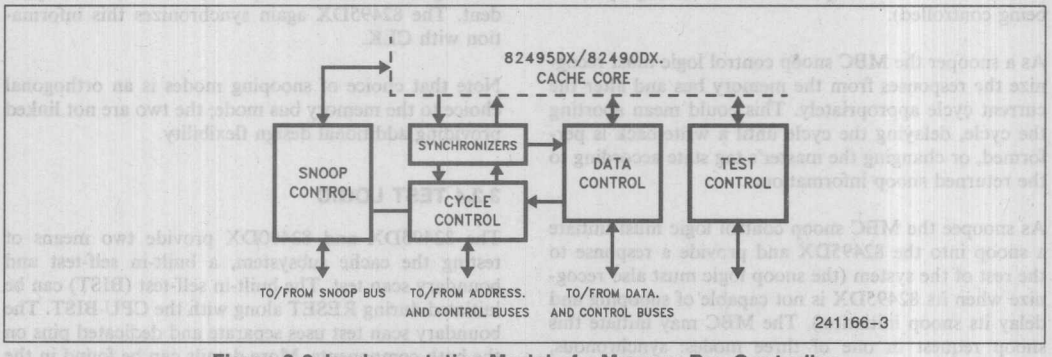


Figure 3-2. Implementation Model of a Memory Bus Controller

Table 3-1. Comparing Clocked and Strobed Memory Bus Implementations

	Advantages	Disadvantages
Clocked	Design techniques for clocked systems are well known.	Must round-up delays to MCLK period quanta. ie 30nsec means two 25nsec MCLKs.
	Fast Arbitration using MCLK state machines.	Some cache-to-cache signals must be synchronized twice, at sender & receiver.
	Burst transfers proceed at MCLK rate.	Memory bus length is limited.
		MCLK skew must be controlled.
Strobed	Delays determined by device speed and physics, not by MCLK quanta.	System design may require delay lines or non-conventional design techniques.
	Each signal only passes through synchronizer once, at the receiver.	Arbitration slow because signals must be synchronized at the arbitor.
	Fewer limits on bus length.	Burst throughput is slowed if each transfer requires acknowledgement from receiver.
	Fewer clock skew worries.	

3.2.3 Snoop Control Block

There are two functions within the snoop control block. When the MBC is master on the memory bus it must snoop all other cache controllers - in this mode it is a snooper (similar to an employer, i.e. in control); when another master has control of the memory bus the snoop control block must respond to a snoop request - in this mode it is a snoopee (similar to an employee, i.e. being controlled).

As a snooper the MBC snoop control logic must recognize the responses from the memory bus and alter the current cycle appropriately. This could mean aborting the cycle, delaying the cycle until a write-back is performed, or changing the master's tag state according to the returned snoop information.

As snoopee the MBC snoop control logic must initiate a snoop into the 82495DX and provide a response to the rest of the system (the snoop logic must also recognize when its 82495DX is not capable of snooping and delay its snoop initiation). The MBC may initiate this snoop request in one of three modes: synchronous, clocked, and strobed. The snoop response of the 82495DX is always synchronous to the CPU CLK.

When initiating the snoop in synchronous snoop mode, all snoop information is latched by the 82495DX synchronous to the CPU CLK. The snoop is then performed on the next CLK edge and the response given on the CLK edge after that. This is the fastest possible method of snooping.

In clocked snooping mode, information is latched by the 82495DX with respect to an external snoop clock source (which is typically slower than CLK). The 82495DX will internally synchronize this information to CLK and provide a response.

In strobed snooping mode, information is latched into the 82495DX with respect to the falling edge of a reference signal. Thus, the snoop initiation is clock independent. The 82495DX again synchronizes this information with CLK.

Note that choice of snooping modes is an orthogonal choice to the memory bus mode; the two are not linked providing additional design flexibility.

3.2.4 TEST LOGIC

The 82495DX and 82490DX provide two means of testing the cache subsystem, a built-in self-test and boundary scan test. The built-in self-test (BIST) can be initiated during RESET along with the CPU BIST. The boundary scan test uses separate and dedicated pins on the both components. More details can be found in the Intel486 DX CPU-Cache module hardware reference manual .

4.0 DECISION PROCESS

This section reviews the key decision criteria that should be considered during the design of an MBC.

Certain choices will limit the options on later choices so it may be necessary to use an iterative approach to reach a final implementation plan. We shall start from the center of the design and work outwards.

4.1 Range of Memory Bus Implementations

The MBC interface was defined with a minimal set of memory bus implementation assumptions in order to maximize design choices. Table 4-1 lists a wide range of implementations possible using the 82495DX cache controller and 82490DX cache SRAMs.

For desktop PCs, a 32-bit simple memory bus is adequate. For a server, workstation or small multiprocessor of two CPUs, a faster 64-bit bus may be required to give adequate bandwidth for graphics frame buffers and intensive numeric calculations. Bus bandwidth requirements grow as the MIPS rating of each CPU in a system grows; for example, a bus adequate for twelve Intel386™ DX CPUs may be too slow for six Intel486™ DX CPUs, as they process far more data per second.

A large multiprocessor system of six or more CPUs may need a wide and fast bus such as Futurebus+, with split-transaction capability to prevent bus bottlenecks from slowing the performance of every processor. Hierarchies of buses and caches can further allow more CPUs with reasonable performance increases as CPUs are added. Compatibility with existing buses is often crucial in product design, so that new faster components can plug into existing machines and I/O devices. Intel's flexible cache subsystem bus interface allows compatibility as well as extension. For proprietary buses, the "proprietor" can design an ASIC or EPLD based MBC incorporating the required features.

4.2 Choosing a Memory Bus

The first stage in the definition of an MBC is the choice of the memory bus. A short review of Table 4-1 should give a first approximation to your memory bus feature set. There are many degrees of freedom possible and an overriding consideration will be the best price/performance for your target application.

For this design example a well-known memory bus is chosen - the Intel486 DX CPU bus. The design will include a 50MHz Intel486 DX CPU-Cache module and MBC which can plug into the Intel486 DX socket of an existing host system, as shown in Figure 4-1 — this effectively produces a Intel486 DX 'super' CPU module with a very large internal cache.

FEATURE	UNIPROC	SMALL 2-3 CPUs	MEDIUM 4-8 CPUs	LARGE 8+ CPUs
		← MULTIPROCESSOR →		
EXAMPLE	PC	COMPAQ SYSTEMPRO NCR3450	COROLLARY 486/SMP NCR3550	SEQUENT SYMMETRY ALLIANT FX2800 NCR 3600
MEMORY BUS				
-STYLE	SIMPLE	← PIPELINED →		CROSSBAR
-WIDTH	32-BIT	32 OR 64	64 OR 128	64 OR 128
-FREQUENCY	20-40 MHz	← 33 MHz OR MORE →		
CACHE	WRITE-THRU	← WRITE BACK →		
ARBITRATION	CENTRAL	CENTRAL	DISTRIBUTED BUS PARKING	DISTRIBUTED BUS PARKING
ERROR DETECT	PARITY	PARITY	ECC	ECC
OTHER FEATURES			SPLIT TRANSACTIONS READ-FOR-OWNERSHIP CACHE-TO-CACHE TRANSFERS EXTERNAL TAGs THIRD LEVEL CACHE	

241166-4

Table 4-1. A Wide Range of Memory Bus Solutions are Possible

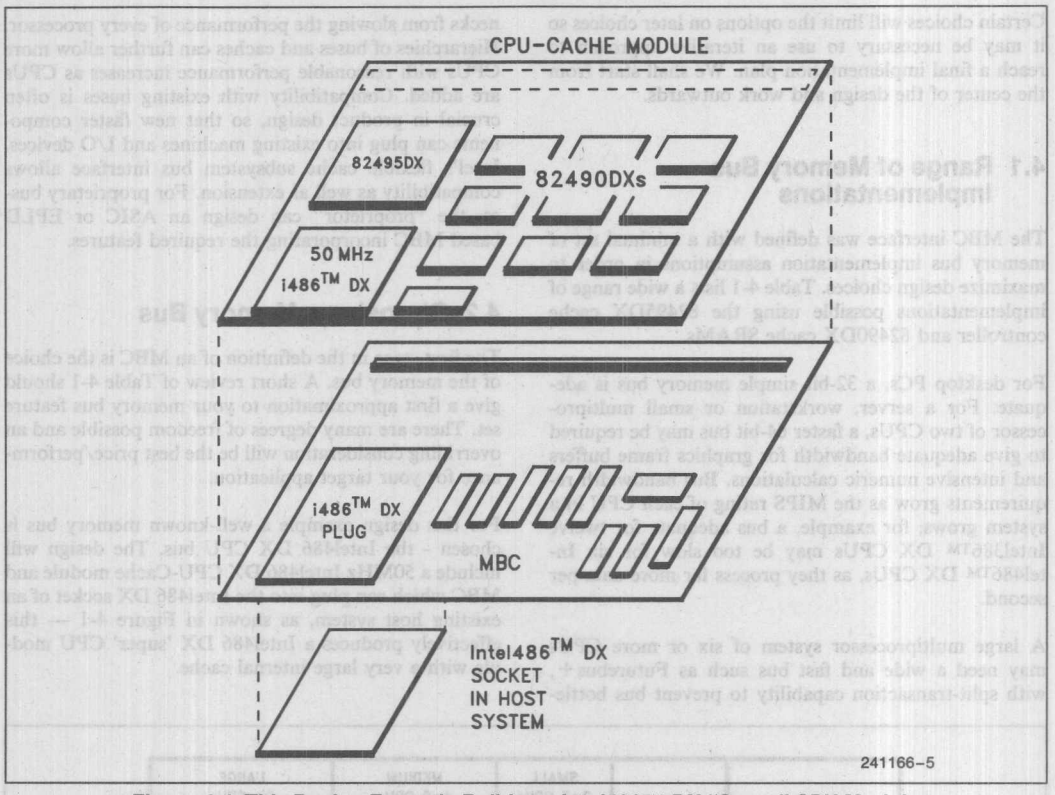
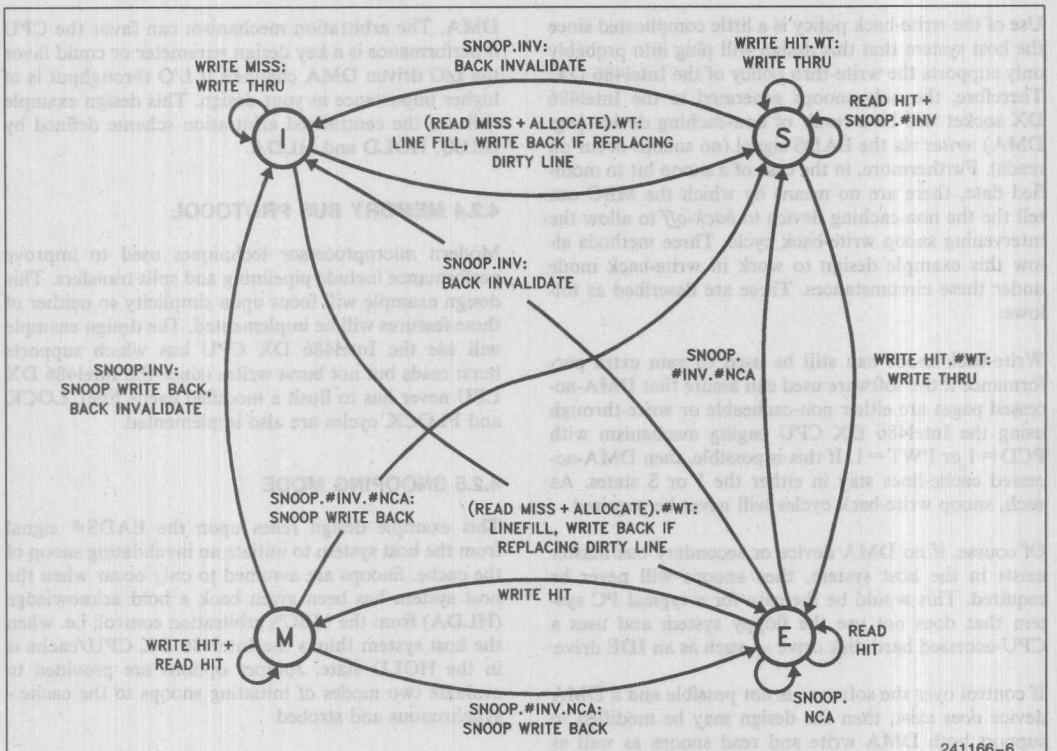


Figure 4-1. This Design Example Builds an Intel486™ DX "Super" CPU Module

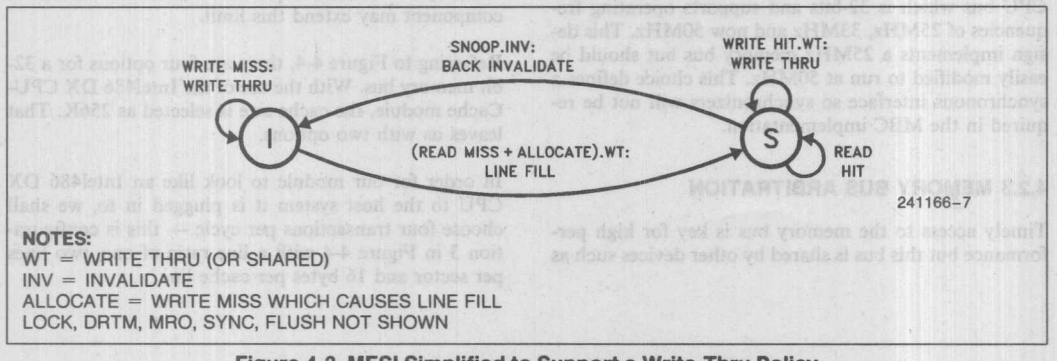
4.2.1 CACHE PROTOCOL

The 8K-bytes of internal Intel486 DX cache supports a write-through memory update policy. The 82495DX/82490DX second level cache solution can support the full MESI cache protocol (see Figure 4-2) — which can be used for either a write-back or write-thru memory update policy. This design example will implement both the write-back and write-thru policies and these will be selectable via a jumper. When write-thru is selected, the cache-line state transitions simplify to that shown in Figure 4-3.



NOTES:
WT = WRITE THRU (OR SHARED)
INV = INVALIDATE
NCA = NON CACHING ACCESS
ALLOCATE = WRITE MISS WHICH CAUSES LINE FILL
LOCK, DRTM, MRO, SYNC, FLUSH NOT SHOWN

Figure 4-2. MESI State Transitions



NOTES:
WT = WRITE THRU (OR SHARED)
INV = INVALIDATE
ALLOCATE = WRITE MISS WHICH CAUSES LINE FILL
LOCK, DRTM, MRO, SYNC, FLUSH NOT SHOWN

Figure 4-3. MESI Simplified to Support a Write-Thru Policy

Use of the write-back policy is a little complicated since the host system that this design will plug into probably only supports the write-thru policy of the Intel486 DX. Therefore, the only snoops generated to the Intel486 DX socket will be a result of non-caching device (e.g. DMA) *writes* via the EADS signal (no snoops occur on reads). Furthermore, in the case of a snoop hit to modified data, there are no means by which the MBC can tell the the non-caching device to *back-off* to allow the intervening snoop write-back cycle. Three methods allow this example design to work in write-back mode under these circumstances. These are described as follows:

Write-back mode can still be used to gain extra performance if the software used can assure that DMA-accessed pages are either non-cacheable or write-through using the Intel486 DX CPU paging mechanism with PCD = 1 or PWT = 1. If this is possible, then DMA-accessed cache-lines stay in either the I or S states. As such, snoop write-back cycles will never be required.

Of course, if no DMA device or secondary bus master exists in the host system, then snoops will never be required. This would be the case for a typical PC system that does not use the floppy system and uses a CPU-accessed hard disk drive — such as an IDE drive.

If control over the software is not possible and a DMA device *does* exist, then the design may be modified to support both DMA write and read snoops as well as snoop write-back cycles. This is described briefly in section 5.3.5.2.

4.2.2 MEMORY BUS WIDTH AND SPEED

A wider, faster memory bus can offer some performance gains but at the cost of some implementation expense. This design example will use the Intel486 DX CPU bus which is 32-bits and supports operating frequencies of 25MHz, 33MHz and now 50MHz. This design implements a 25MHz memory bus but should be easily modified to run at 50MHz. This choice defines a synchronous interface so synchronizers will not be required in the MBC implementation.

4.2.3 MEMORY BUS ARBITRATION

Timely access to the memory bus is key for high performance but this bus is shared by other devices such as DMA. The arbitration mechanism can favor the CPU if performance is a key design parameter or could favor the I/O driven DMA channels if I/O throughput is of higher importance in your design. This design example will use the centralized arbitration scheme defined by BREQ, HOLD and HLDA.

4.2.4 MEMORY BUS PROTOCOL

Modern microprocessor techniques used to improve performance include pipelining and split transfers. This design example will focus upon simplicity so neither of these features will be implemented. The design example will use the Intel486 DX CPU bus which supports burst reads but not burst writes (since the Intel486 DX CPU never has to flush a modified cache line). LOCK and PLOCK cycles are also implemented.

4.2.5 SNOOPING MODE

This example design relies upon the EADS# signal from the host system to initiate an invalidating snoop of the cache. Snoops are assumed to only occur when the host system has been given back a hold acknowledge (HLDA) from the MBC's arbitration control; i.e. when the host system thinks the Intel486 DX CPU/cache is in the HOLD state. Jumper options are provided to evaluate two modes of initiating snoops to the cache - synchronous and strobed.

4.3 Choosing Cache Parameters

The choice of memory bus width will limit the choice of second level cache sizes. If the cache size you had planned is not available then the memory bus width decision should be re-evaluated. The current 82490DX cache SRAM supports cache sizes of 128K, 256K and 512K-bytes — future implementations of the 82490DX component may extend this limit.

Referring to Figure 4-4, there are four options for a 32-bit memory bus. With the use of the Intel486 DX CPU-Cache module, the cache size is selected as 256K. That leaves us with two options.

In order for our module to look like an Intel486 DX CPU to the host system it is plugged in to, we shall choose four transactions per cycle — this is configuration 3 in Figure 4-4 with a line ratio of one, two lines per sector and 16 bytes per cache line.

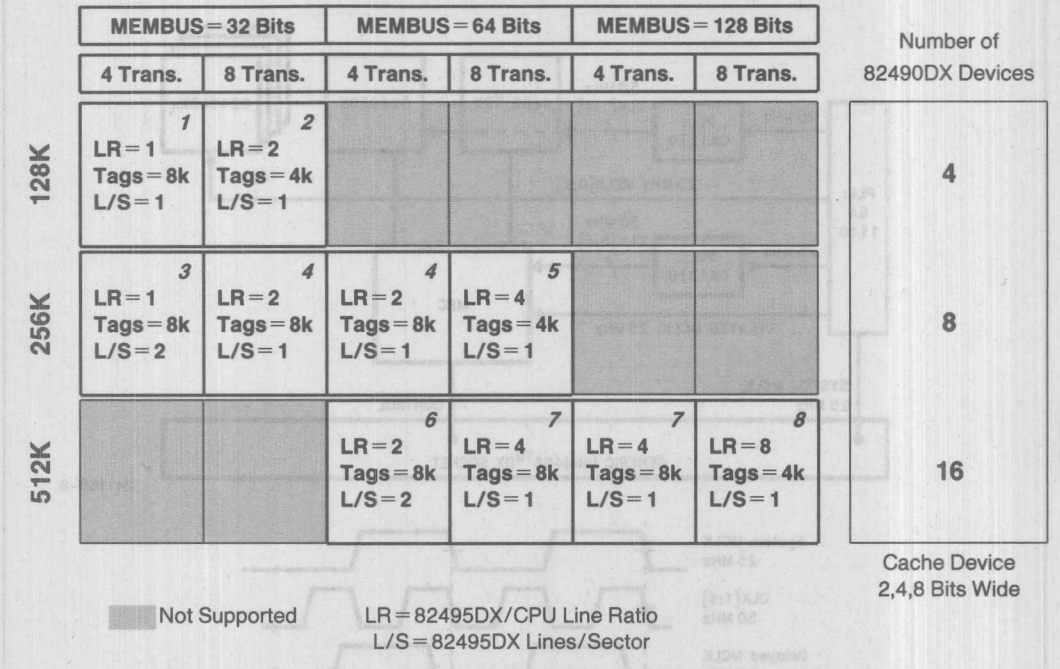


Figure 4-4. The Choice of Memory Bus Width and Cache Size are Coupled

4.4 Other Assumptions

In order to simplify the example design, some additional assumptions were made regarding the host system:

— It is assumed that the host system has no parity — DP0:3 and PCHK# are not used.

— All transfers on the memory bus are assumed to be to 32-bit devices — BS8# and BS16# are no connects.

— BOFF# is assumed to be a no-connect on the host system.

— MKEN# from the host system is ignored. The cacheability attribute is generated by a decoder in the MBC.

Of course, the reader is free to modify the MBC example design described here to support the functionality required by the signals above.

Finally, use of the boundary scan logic (TDI, TDO, TCK, TMS pins) is not implemented.

5.0 DESIGN PROCESS

Now that the system design decisions have been made, this section details the logic design process for the MBC.

5.1 Clock Generation

As discussed earlier, this design implements a 25MHz memory bus. This choice is called a divided synchronous clock scheme and synchronizers are not required. The timing reference is derived from the 25MHz clock input of the Intel486 DX CPU socket. One Gazelle GA1110 and two Gazelle GA1210 phase locked loop (PLL) clock doubler chips are used to derive 50MHz reference clocks: one for the CPU-Cache Module and one for the MBC PLDs (refer to Figure 5-1). The GA1210 is able to provide the four 50MHz clocks into the CPU-Cache module with a maximum output-to-output skew of 500ps.

All of the MBC's state machines operate at 50MHz. In order to interface to the 25MHz host system's Intel486 DX interface, a delayed version of the 25MHz clock is generated by the GA1110 in Figure 5-1: a delayed MCLK. This signal is used as an input to the MBC state machines as a qualifier for signals to and from the host interface. This MCLK scheme assures that the MBC outputs mimic a 25MHz Intel486 DX CPU inter-

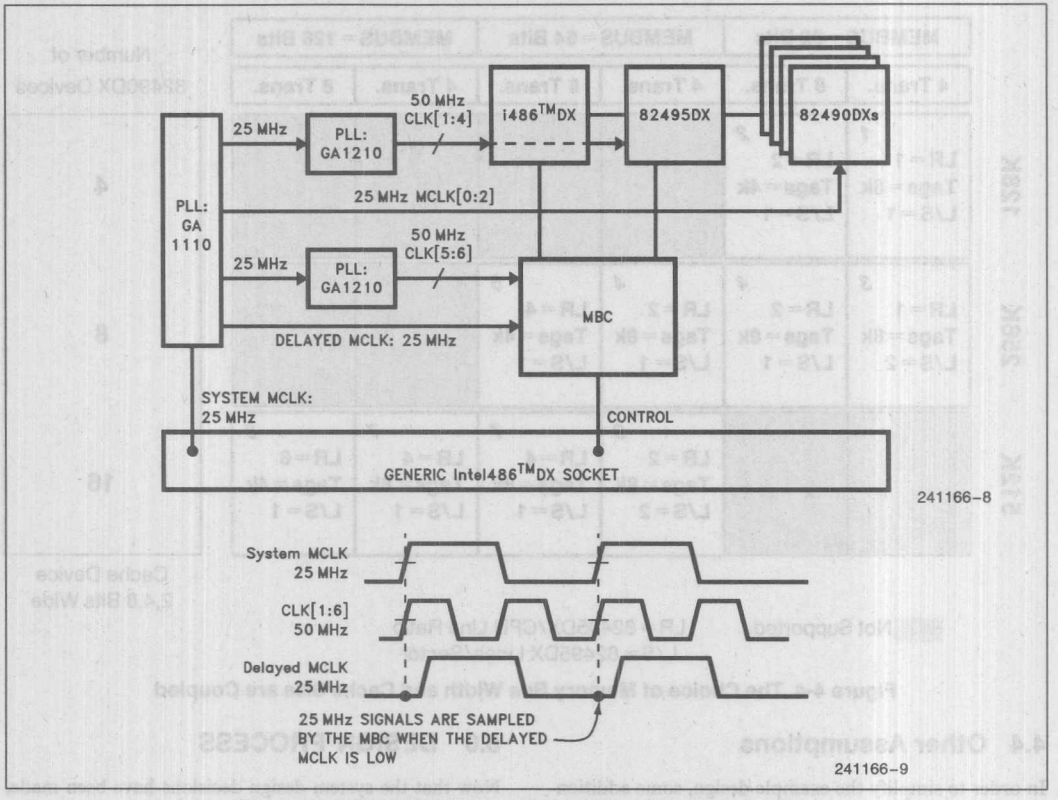


Figure 5-1. The MBC Uses a Divided Synchronous Design

face. It is used to maintain the MBC outputs active for two CLKs and helps to correctly sample the memory bus inputs to the MBC, such as MBRDY# (so they are not sampled active twice mistakenly).

Finally, note that the use of PLLs is only required because of the need to align the 50MHz clocks with the host system's clock. If the clocks are generated by the MBC and supplied to the host system instead, simpler low-skew clock buffers may suffice.

5.2 Reset and Initialization

The configuration options of the cache subsystem are selected by driving certain inputs of the 82495DX and 82490DX during RESET. Note that many of the inputs are configuration inputs only during RESET and are used for operational inputs or outputs following RESET. An example of this is the KWEND# [CFG2] in-

put to the 82495DX that is forced active during reset by the XWND state machine but is used to indicate *cacheability window end* during normal operation. The configuration inputs used during reset are tabulated in the Appendix.

Table 5-1 lists the configuration options for this design. These signals are used to configure different modes during operation. Most of the jumper options included here are intended for evaluation purposes. They may be removed if so desired.

5.3 Logic Design

The operation of the CPU-Cache subsystem's interface is well defined (refer to Section 9) and since the memory bus is an Intel486 DX bus then this too is well defined. The logic task then, entails matching the protocols of the two bus specifications.

Signal names as defined in the data sheets for the Intel486 DX CPU, 82495DX cache controller and 82490DX SRAMs will be used throughout this discussion (refer to Section 8.0 for additional information). Signals from the memory bus are prefixed with an 'M'.

In the illustrated state diagrams, the following convention is used: During transitions, a hash (#) following a signal name indicates that the signal is not asserted; and a signal name without a hash indicates that the signal is asserted (e.g. BGT means that BGT# = 0). Signals that appear inside states bubbles indicate that the signal is asserted (e.g. BGT inside a bubble means that BGT# = 0 in that state).

The delayed MCLK used by the MBC state machines is referred to as simply MCLK in the state machine descriptions.

Table 5-1. Configuration Options

Signal	Jumper	Used by:	Explanation (jumper setting)	Normal operation
MWBWT#	J6	82495DX	Write-thru mode (if installed) Write-back mode (if open)	Installed
DRCTM#	J7	82495DX	Enable direct-to-modified transition (if installed)	Open
SLFTST#	J2	XCRDY state machine	Enables CPU-Cache self test (installed)	Open
SNPDIS#	J11	Snoop state machine	Disable snoops (installed)	Open
SNPNCA	J5	82495DX	Non caching snoops (open)	Open
ENKEN#	J3	Address Decoder	Enable caching (installed)	Installed
EPKEN#	J8	Address Decoder	Enable BIOS caching (installed)	Installed
RFO#	J10	82495DX	Enable Read for ownership (jumper installed)	Open
CPLOCKEN	J13	State machine	Enable PLOCK cycles (open)	Open
ENALLC#	J4	Address Decoder	Enable allocations (installed)	Installed
SNPCLK	J14	82495DX	Select synchronous (installed) or strobed (open) snooping modes	Open
FLUSH#	J12	82495DX	Enable flushing (installed)	Installed
EWRMRST#	J9	XWRMRST state machine	Select warm reset option (see section 5.3.9)	Open

5.3.1 82495DX CYCLES VERSUS Intel486™ DX CYCLES

The 82495DX cache controller drives a sequence of output signals and expects a sequence of cycle progress signals as shown in Figure 5-2.

CADS# indicates the start of the cycle address phase. CDTS# tracks CADS# and indicates the start of the cycle data phase. For READ cycles CDTS# indicates that the CPU data bus is in read mode and under the control of the MBC from the next CLK until CRDY#. It indicates to the MBC the earliest time that BRDY# can be returned to the 82495DX. For Write cycles CDTS# indicates that the first piece of data is available on the memory bus.

At the beginning of the cycle, cycle information signals are presented to the MBC that indicate to the MBC the type and the length of the cycle. These signals along with the type of ready that is returned from the host system (RDY# or BRDY#) tell the MBC what type of cycle to run as shown in Table 5-2.

To handle the various types and phases of a 82495DX cycle, we start by partitioning the MBC into several state machines as shown in Figure 5-3. State machines XMBREQ and XHLDA handle the cycle requests and host system arbitration. State machine XMADS generates BGT# and the Intel486 DX signal MADS# to the host system once bus ownership is granted. XMBLAST generates the signal MBLAST# as well as MA[2:3] to the host system. This is required to convert multiple dword requests from the 82495DX into Intel486 DX burst cycles or multiple single cycles. For example, a write-back cycle of 16 bytes needs to be separated into four separate dword write cycles to the host system.

State machine XWND and a decoder provide KWEND#, SWEND# and cacheability information to the 82495DX. Locked and psuedo-locked cycles are handled by XMLOCK and XMPLOCK. Finally, XBRDY, XDISBRDY and XCRDY take care of the generation of BRDY# and CRDY#.

These state machines will be built up step-by-step as we progress through the cycles - the completed and final state machines are covered in Section 7.

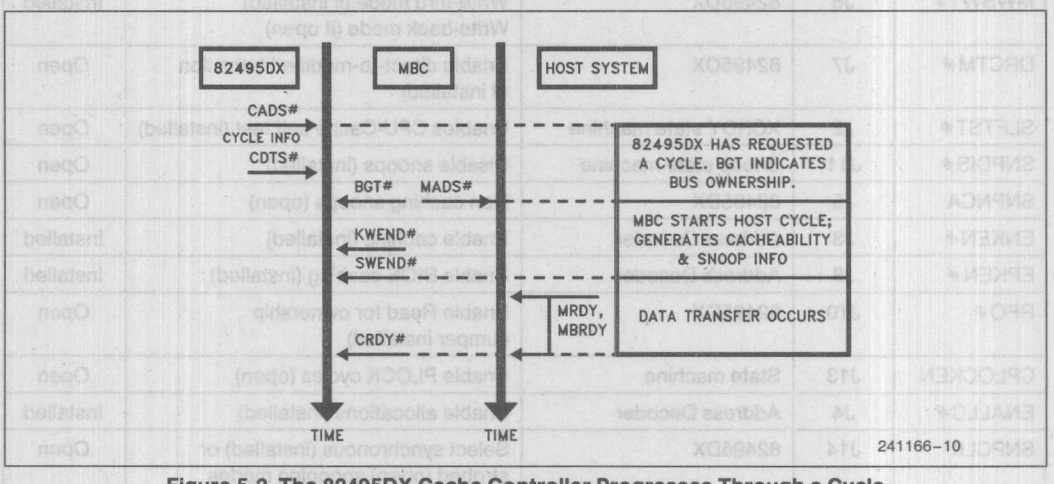


Figure 5-2. The 82495DX Cache Controller Progresses Through a Cycle

Table 5-2. MBC Cycle Translations

SIGNALS THAT DETERMINE MBC CYCLES	
Inputs from the 82495DX	**Inputs from the Intel486™DX Host System**
CWR#, CMIO#, MCACHE#, CLEN[0:1],RDYSRC, MKEN# (from decode of 82495DX addresses), KLOCK#,CPLOCK#	MRDY#, MBRDY#
POSSIBLE 82495DX CYCLES	RESULTANT Intel486™DX CYCLES
Posted Write	Single Cycle Write
Non-cacheable Read[1] (one dword)	Single Cycle Read
Non-cacheable Read[1] (2 or 4 dwords)	Burst Read or Two or Four Single Cycle Reads
Line Fill	Burst Read or Four Single Cycle Reads
Allocation	Burst Read or Four Single Cycle Reads
Write Back	Burst Write[2] or Four Single Cycle Writes
I/O Cycle	I/O Cycle
Lock Read	Lock Read
Lock Write	Lock Write

[1] A non-cacheable read occurs when MCACHE# or MKEN# = 1.
[2] The MBC will support burst writes even though the host system does not.

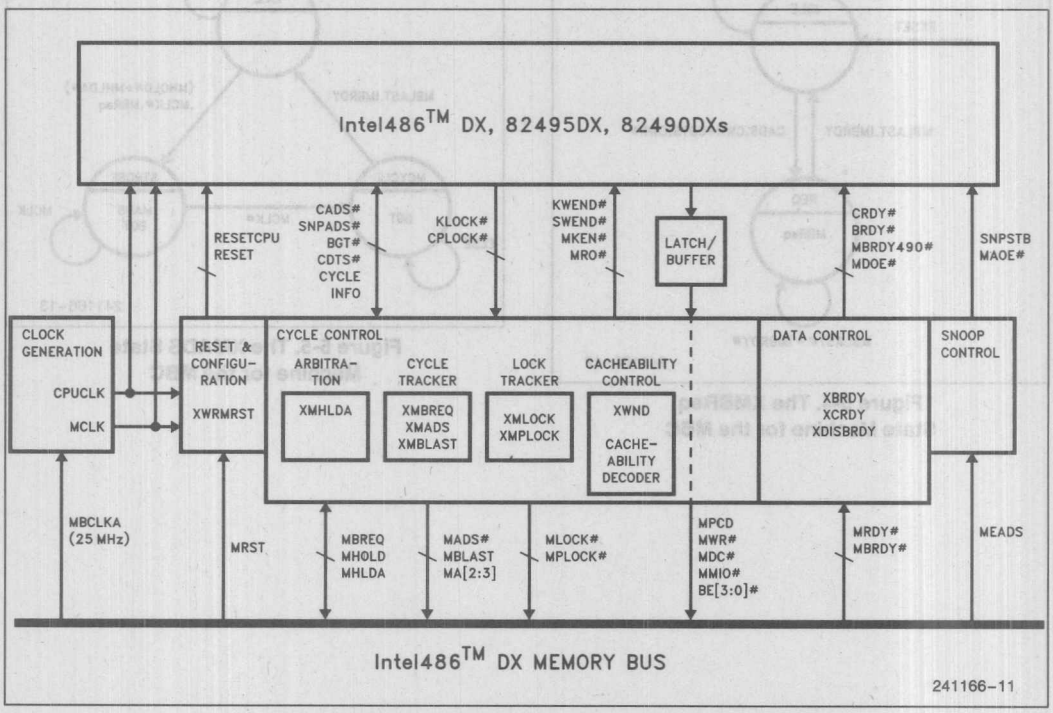


Figure 5-3. Block Diagram of the Intel486™ DX Simple MBC

5.3.2 REQUESTS FOR THE MEMORY BUS

We can define an MBC stae REQ (Memory Bus Request) entered due to a write request (CDTS.CWR#) or a read request (CADS.CWR) as shown in Figure 5-4. Entry into this state causes the MBREQ signal to the host system to be asserted. The assertion of the signal CDTS# is used to qualify writes so that the translation into REQ is independent of whether it was started by CADS# or SNPADS#. If the 82495DX is not pipelined, CDTS# is always asserted along with CADS#. For snoop write-back cycles, CDTS# lags SNPADS# by two to four CLK cycles.

MBREQ is also used by other MBC state machines. We shall retain MBREQ active until the cycle is completed as indicated by MBLAST# active (MBLAST# indicates the last transfer in a cycle and is described in section 5.3.3.3). An internal signal, IMBRDY, equal to (MRDY + MBRDY).MCLK# is useful to simplify the state machines.

5.3.3 STARTING A CYCLE

This design example uses a HOLD/HLDA arbitration scheme (state machine described later). If a hold is not being requested (MHOLD = 0) or not being granted (MHLDA = 0) to the Intel486 DX socket then the MBC grants the cycle to the 82495DX by driving BGT# active as shown in Figure 5-5. An address strobe is also generated (MADS = 1) onto the memory bus at this time. Note that MADS# is active for one MCLK period. BGT# is maintained active until the cycle completes as indicated by MBLAST# active.

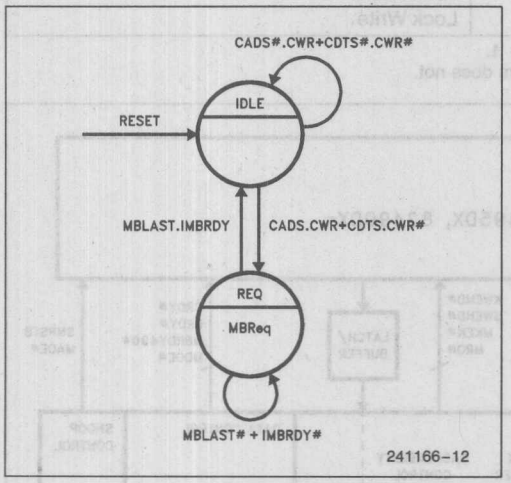


Figure 5-4. The XMBReq State Machine for the MBC

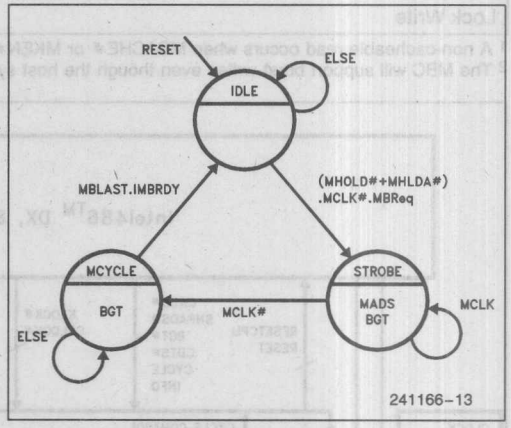


Figure 5-5. The XMADS State Machine for the MBC

The signal BGT# is also connected to the MSEL#[MTR4/MTR8#] input of the 82490DXs.

If the CPU is not in HOLD, the cycle may be started in the same cycle as MBReq - this can save a clock cycle and will result in higher performance. To accomplish this, the transition out of the IDLE state of XMADS could be augmented to include this parallel operation: the additional term would be MHOLD#.MHLDA#.MCLK#.(CADS.CWR + CDTS.CWR#). This additional term is added to XMADS in a later Figure.

5.3.4 COMPLETING THE CYCLE

The MBC responds to the CPU generated address by responding with cycle progress signals and then controls the data flow into/out of the 82490DX cache SRAMs.

5.3.4.1 Controlling Cacheability

The MBC closes the cacheability window by driving KWEND# after BGT# is active for two CLKs as shown in Figure 5-6. At this point, the 82495DX will sample MKEN# and MRO#; these signals indicate the cacheability and read-only attributes of the addressed memory cycle. These attributes can be determined by decoding the current address. Based on those attributes the 82495DX executes allocations, line-fills, replacements, etc. From Figure 5-6, the time available to decode MKEN# and MRO# from the address bus is 63 ns (not including clock skew).

The MBC will also close the snoop window by driving SWEND# at which point the 82495DX will sample MWB/WT# and DRCTM#; these signals are determined by snooping the other caches in the system in a multiprocessor system. At this point the 82495DX updates its TAGRAM state related to the line access in progress. However, since there are no other caches to snoop in this design example, SWEND# may be asserted at the same time as KWEND#. Cache line attribute information such as MWB/WT# is sampled according to the fixed jumper settings.

The state machines for KWEND# and SWEND# are shown in Figure 5-6. (In the schematics, the KWEND# and SWEND# inputs of the CPU-Cache module are driven by a single signal SKWEND#.)

An address decoder PLD is used to generate MKEN# and MRO# for the 82495DX. This design example includes PLD equations for the decoder corresponding to the cacheable areas of a typical PC address map. The MRO (read-only) attribute is required when caching the BIOS for proper operation in a PC system. MRO cache lines are cached in the 82495DX/82490DX but not in the Intel486 DX CPU. CPU writes to MRO protected cache lines in the 82495DX/82490DX cause a write thru cycle.

The format of the MKEN# decode equation is as follows:

```
MKEN = ENKEN.[Cacheable addr + BIOS addr.EPKEN].
          [CW/R                          (read accesses)
           + CW/R#.ENALLC.PALLC]         (write allocations)
```

where ENKEN enables caching, EPKEN enables BIOS caching, and ENALLC enables write allocations. Note that MKEN# must be returned active during cache memory write misses in order for the 82495DX to generate an allocation cycle.

5.3.4.2 Supporting Multiple Transfer Requirements

When CADS# is generated, the 82495DX will output CLEN0, CLEN1, CW/R#, and MCACHE#. These signals provide the MBC with enough information to determine the cycle length. CLEN0 and CLEN1 indicate the number of transfers (one, two or four) that the CPU expects if the cycle is returned non-cacheable. MCACHE# is driven as an indication that the cache controller could cache this access; the MBC can make the line cacheable by returning MKEN# active. The possible cycles were shown in Table 5-2.

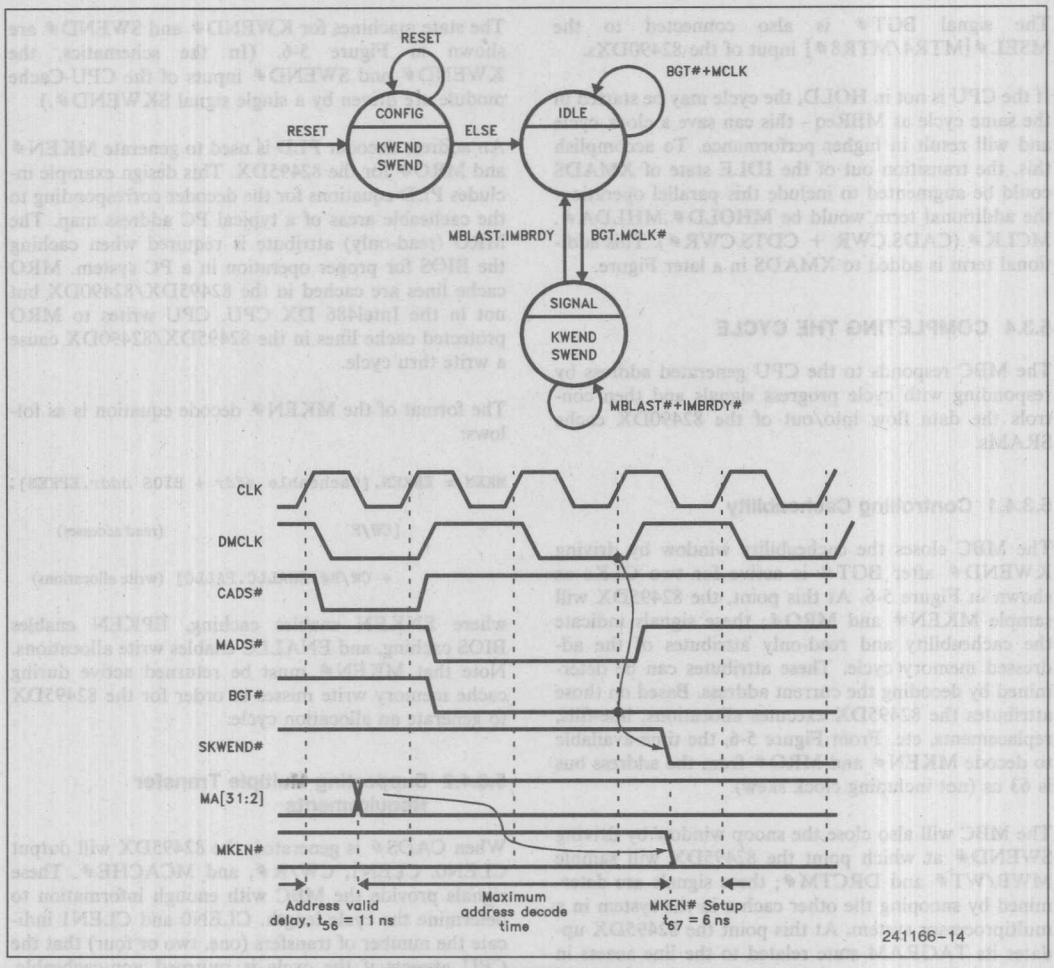
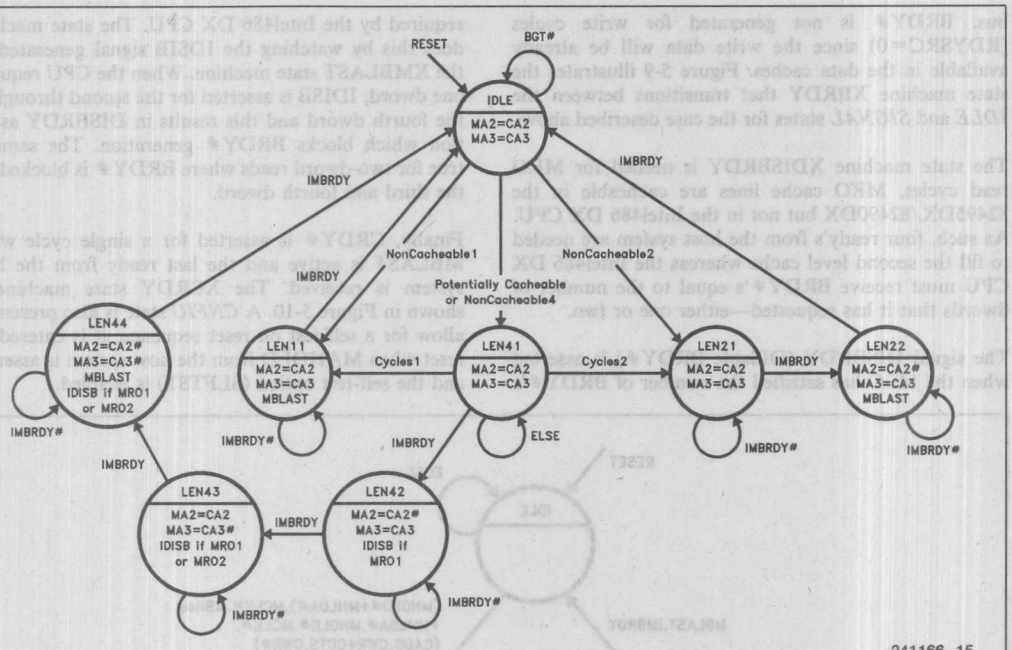


Figure 5-6. KWEND and SWEND are Generated by the XWND State Machine on the MCLK Following BGT

The XMBLAST state machine (Figure 5-7) is used to determine the number of transfers required by decoding the cycle information from the 82495DX. Using this information, it generates the lower order burst addresses (MA2, MA3) to the host system in the appropriate order after each transfer. It also generates the signal **MBLAST#**—this is driven into the host system and also used internally by other MBC state machines to indicate the end of the cycle.

Finally, the XMBLAST state machine generates the signal IDISB (Internal Disable BRDY#). This signal is used during read-only cache linefills by the state machine that generates BRDY# (discussed later). It is asserted during cacheable cycles in states LEN42 through LEN44 for a one-dword read and in states LEN43 through LEN44 for a two-dword read.

In the case where the 82495DX has requested more than one transfer (two or four dwords) and the host system returns MRDY# instead of MBRDY#, the cycle cannot be burst. Instead, the cycle is broken down into multiple single cycles to the host system. This will be the case for write-back cycles (since the host system probably does not support burst writes) and for code fetches from non-burstable BIOS EPROMs.



NonCacheable1 = BGT.MCACHE#.CLEN1#.CLEN0#
NonCacheable2 = BGT.MCACHE#.CLEN1#.CLEN0
Potentially Cacheable or NonCacheable4 = BGT.(MCACHE+CLEN1)
Cycles1 = KWEND.MKEN#.CLEN1#.CLEN0#.IMBRDY#
Cycles2 = KWEND.MKEN#.CLEN1#.CLEN0.IMBRDY#
MRO1 = MKEN.MCACHE.CLEN1#.CLEN0#.IMBRDY
MRO2 = MKEN.MCACHE.CLEN1#.CLEN0.IMBRDY

NOTE:
State LEN21 refers to the first cycle of a two cycle transfer. Similarly for other state names.

Figure 5-7. The XMBLAST State Machine Tracks Requests for Multiple Dwords

The XMADS state machine, introduced in Figure 5-5 must be augmented to produce multiple memory address strobes when MRDY# is returned before the cycle is over (see Figure 5-8). Also, the additional term that enables quicker transitions out of *IDLE* is added.

5.3.4.3 Controlling the Data Flow

The MBC subsection that controls data flow must provide the following functionality:

— control the data into and out of the 82490DX data caches via MBRDY490 and MDOE#

— tell the 82490DXs when the data transfer is over via MEOC#

— tell the Intel486 DX when to transfer data to and from the CPU's data bus via BRDY#

— tell the 82495DX when the cycle is complete via CRDY#.

Data is transfered into or out of the 82490DXs whenever the host system tells the MBC that it has or can accept data. Thus, the generation of MBRDY490# is straightforward:

MBRDY490 = (MRDY+MBRDY).MCLK#.

The output enable signal for the data caches MDOE# is simply equal to BGT.CWR#. The generation of MEOC# is also simple:

MEOC = (MRDY+MBRDY).MCLK#.MBLAST

where MBLAST is provided by the XMBLAST state machine.

Under most conditions, the generation of BRDY# is also straightforward. BRDY# is asserted for all reads except allocation cycles (RDYSRC=1) and tracks MRDY# or MBRDY# from the host system by one MCLK period. This allows enough time for the data to propagate through the 82490DXs from the memory

bus. BRDY# is not generated for write cycles (RDYSRC=0) since the write data will be already available in the data caches. Figure 5-9 illustrates the state machine XBRDY that transitions between the *IDLE* and *SIGNAL* states for the case described above.

The state machine XDISBRDY is needed for MRO read cycles. MRO cache lines are cacheable in the 82495DX/82490DX but not in the Intel486 DX CPU. As such, four ready's from the host system are needed to fill the second level cache whereas the Intel486 DX CPU must receive BRDY#'s equal to the number of dwords that it has requested—either one or two.

The signal DISBRDY (DISable BRDY#) is asserted when the MBC has satisfied the number of BRDY#'s required by the Intel486 DX CPU. The state machine does this by watching the IDSIB signal generated by the XMBLAST state machine. When the CPU requests one dword, IDISB is asserted for the second through to the fourth dword and this results in DISBRDY assertion which blocks BRDY# generation. The same is true for two-dword reads where BRDY# is blocked for the third and fourth dword.

Finally, CRDY# is asserted for a single cycle when MBLAST is active and the last ready from the host system is received. The XCRDY state machine is shown in Figure 5-10. A *CNFIG* state is also present to allow for a self-test on reset sequence; it is entered on reset when MAHOLD from the host system is asserted and the self-test option (SLFTST) is enabled.

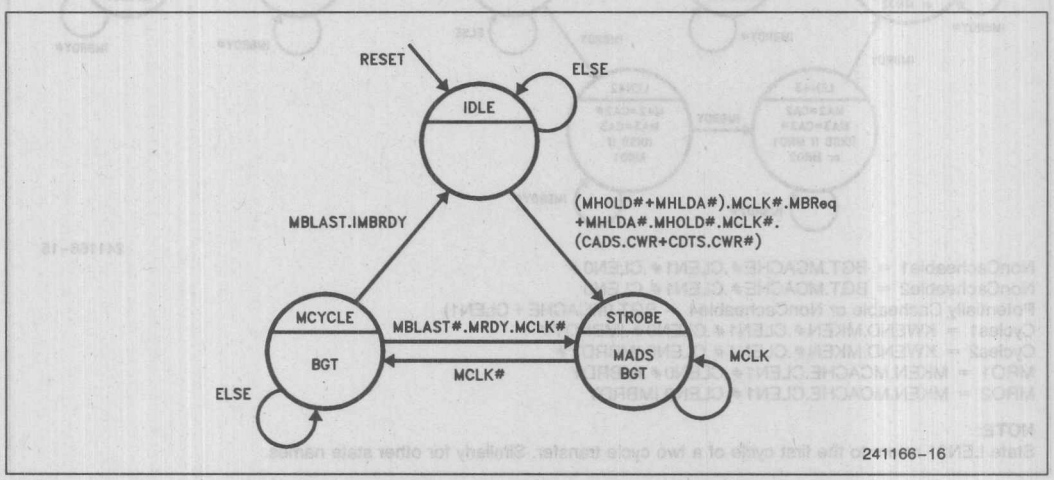


Figure 5-8. The XMADS State Machine Extended to Generate Multiple Address Strobes

5.3.5 SNOOPEE CYCLES

When the MBC does not have control of the memory bus, the host system generates snoops via MEADS# into the CPU-Cache/MBC. In this design example MEADS# is qualified by MCLK# to bring it into the CLK timing domain and generates SNPSTB (SNooP STroBe) into the 82495DX. Note that synchronous and strobed modes of snoop timing can be evaluated with this design.

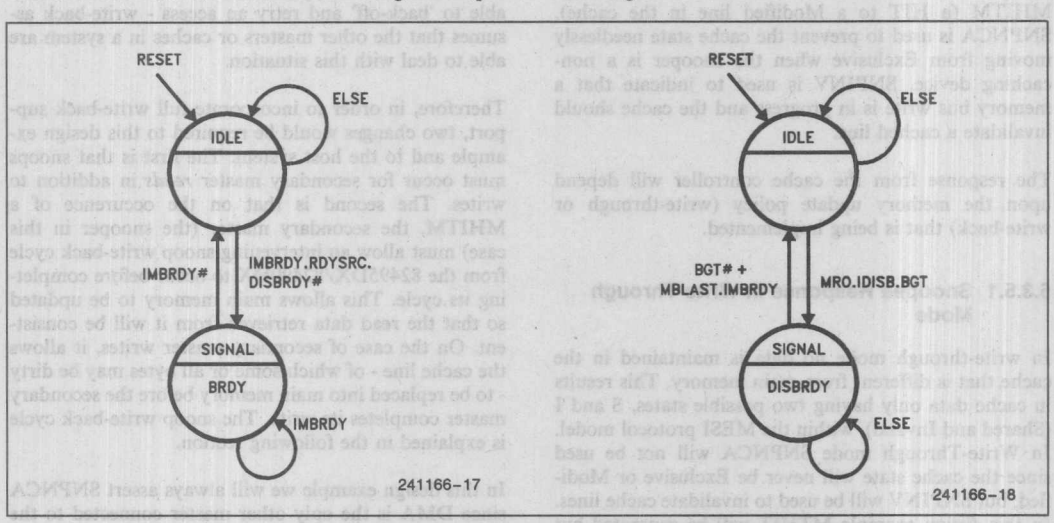


Figure 5-9. Generating BRDY#'s for the CPU-Cache (XBRDY & XDISBRDY)

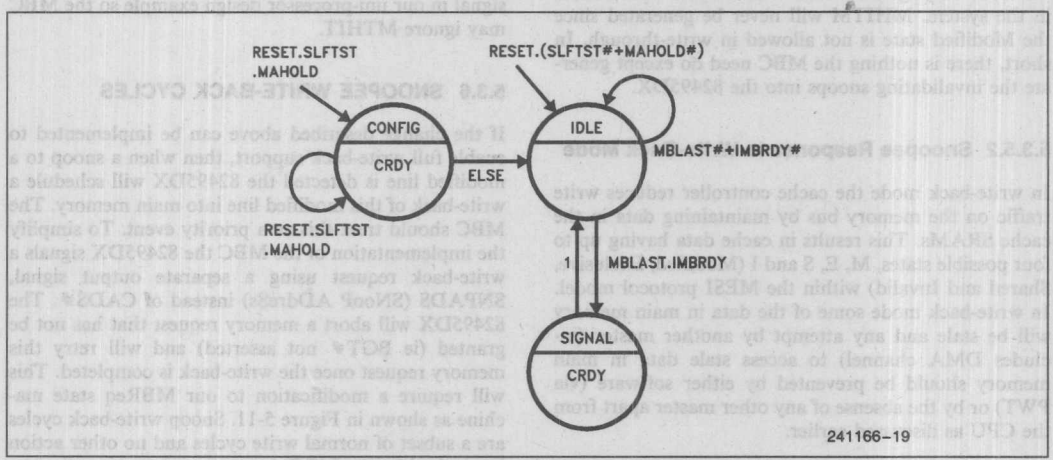


Figure 5-10. Generating Ready's for the Cache Subsystem (XCRDY)

The 82495DX samples the memory address, SNPNCA (SNooP Non-Caching Access) and SNPINV (SNooP INValidate) with SNPSTB and will generate state information indicated by MTHIT (a Tag HIT, ie the snoop address is contained within the cache) and MHITM (a HIT to a Modified line in the cache). SNPNCA is used to prevent the cache state needlessly moving from Exclusive when the snooper is a non-caching device. SNPINV is used to indicate that a memory bus write is in progress and the cache should invalidate a cached line.

The response from the cache controller will depend upon the memory update policy (write-through or write-back) that is being implemented.

5.3.5.1 Snoopee Response in Write-Through Mode

In write-through mode no data is maintained in the cache that is different from main memory. This results in cache data only having two possible states, S and I (Shared and Invalid), within the MESI protocol model. In Write-Through mode SNPNCA will not be used since the cache state will never be Exclusive or Modified; but SNPINV will be used to invalidate cache lines. In this design example MTHIT will be generated but the MBC may ignore it since there are no other caches in the system. MHITM will never be generated since the Modified state is not allowed in write-through. In short, there is nothing the MBC need do except generate the invalidating snoops into the 82495DX.

5.3.5.2 Snoopee Response in Write-Back Mode

In write-back mode the cache controller reduces write traffic on the memory bus by maintaining data in the cache SRAMs. This results in cache data having up to four possible states, M, E, S and I (Modified, Exclusive, Shared and Invalid) within the MESI protocol model. In write-back mode some of the data in main memory will be stale and any attempt by another master (includes DMA channel) to access stale data in main memory should be prevented by either software (via PWT) or by the absense of any other master apart from the CPU as discussed earlier.

If secondary masters cannot be prevented from accessing memory that has been write-back enabled, the 82495DX will assert MHITM when the accessed data is found modified in the cache. The 82495DX/82490DX MUST be allowed to write-back this data to main memory and the other master should be told to retry the access. A 'typical' DMA controller will not be able to 'back-off' and retry an access - write-back assumes that the other masters or caches in a system are able to deal with this situation.

Therefore, in order to incorporate full write-back support, two changes would be required to this design example and to the host system. The first is that snoops must occur for secondary master *reads* in addition to writes. The second is that on the occurence of a MHITM, the secondary master (the snooper in this case) must allow an intervening snoop write-back cycle from the 82495DX/82490DX to occur before completing its cycle. This allows main memory to be updated so that the read data retrieved from it will be consistent. On the case of secondary master writes, it allows the cache line - of which some or all bytes may be dirty - to be replaced into main memory before the secondary master completes its write. The snoop write-back cycle is explained in the following section.

In this design example we will always assert SNPNCA since DMA is the only other master connected to the Intel486 DX CPU socket. There are no other caches to signal in our uni-processor design example so the MBC may ignore MTHIT.

5.3.6 SNOOPEE WRITE-BACK CYCLES

If the change described above can be implemented to enable full write-back support, then when a snoop to a modified line is detected the 82495DX will schedule a write-back of this modified line into main memory. The MBC should treat this as a priority event. To simplify the implementation of the MBC the 82495DX signals a write-back request using a separate output signal, SNPADS (SNooP ADdreSs) instead of CADS#. The 82495DX will abort a memory request that has not be granted (ie BGT# not asserted) and will retry this memory request once the write-back is completed. This will require a modification to our MBReq state machine as shown in Figure 5-11. Snoop write-back cycles are a subset of normal write cycles and no other action is required by the MBC.

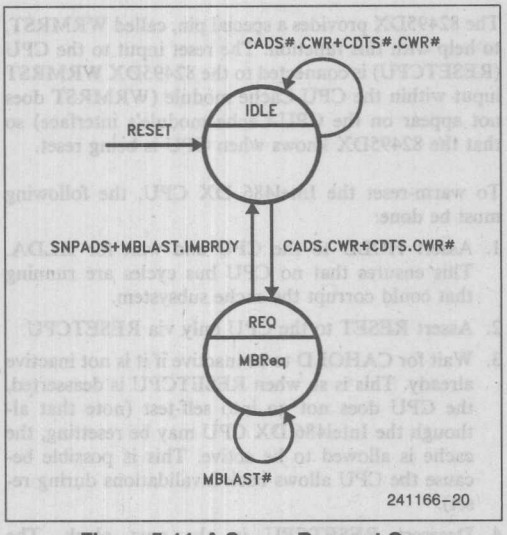


Figure 5-11. A Snoop Request Can Abort a Cycle Request (XMBReq)

5.3.7 SNOOPER REQUIREMENTS

As a snooper, the MBC must generate snoops into other caches in the system during cacheable memory reads and writes. The other caches will respond with both MTHIT and MHITM. The 82495DX uses this information to update the status bits of the accessed line according to the MESI model. Since the host system in this case is assumed to have no other cache apart from the 82495DX/82490DX, no snooper functions are required.

5.3.8 SPECIAL CYCLES

The 82495DX handles all memory accesses and passes non-cachable cycles through to the MBC. The 82495DX will also pass I/O cycles, cache maintenance cycles (such as FLUSH and SYNC) and LOCK/PLOCK cycles through to the MBC.

5.3.8.1 I/O cycles

I/O cycles are decoded by the 82495DX and not posted. These cycles wait until all buffers have been written, and all cycles have been completed, before they cause CADS# assertion. The CPU waits until the I/O ends with the MBC's BRDY# assertion before it continues.

5.3.8.2 FLUSH/SYNC cycles

The on-chip cache of the Intel486 DX CPU can be flushed via the CPU's FLUSHCPU# pin. On the 82495DX/82490DX, there is an equivalent input called PFLUSH# which causes all modified lines to be written back to main memory before the second level cache is invalidated. In this design these two FLUSH inputs are both connected to MFLUSH# coming from the host system.

When the cache subsystem is performing a flush, many write back cycles are required. These cycles look like ordinary write back cycles, and are handled as such. FSIOUT# is active during these write back cycles, so when FSIOUT# goes inactive the cycle is complete.

The Intel486 DX instruction set includes commands that control the on-chip cache. INVD and WBINV both invalidate the on-chip cache. When executed, these commands also appear as external I/O-like special cycles. If so desired, the MBC design may be modified to decode these cycles and to provide the necessary control of the 82495DX/82490DX second-level cache.

SYNC cycles apply only to the second level cache and are similar to FLUSH cycles except that the cache lines are not invalidated upon completion. SYNC cycles are initiated by asserting the SYNC input to the 82495DX. However, this feature is not implemented in this design.

5.3.8.3 LOCK cycles

The 82495DX provides a lock signal for the memory bus called KLOCK#. KLOCK# is generated by the 82495DX whenever the CPU generates the LOCK# signal. KLOCK#, like the other cycle attributes, is valid with CADS# assertion.

When the CPU generates a locked cycle, the 82495DX always generates a bus cycle even if the access was a cache hit. If the access was a cache miss, the locked cycle will be non-cacheable to both the 82495DX and CPU, so the information is passed through the 82495DXs to the CPU with BRDYs generated by the MBC. If the locked read cycle is a modified hit in the 82495DX, the 82495DX ignores the data that it is receiving and supplies data from the cache array (in accordance with BRDYs supplied by the MBC). Locked writes are posted like any other write. Locked cycles, both reads and writes, never change the 82495DX tag state. The state machine for MLOCK# is shown in Figure 5-12. MLOCK# is driven active upon start of a locked cycle.

Following the 82495DX's protocol, KLOCK# may go inactive in the CLK cycle following CADS# assertion. However, the MLOCK# signal must stay active for the whole cycle. Therefore, it is necessary to hold MLOCK# until KLOCK# is deasserted and it is the last transfer in the cycle, or KLOCK# is deasserted and it is the beginning of a new unlocked cycle.

5.3.8.4 PLOCK cycles

The 82495DX supports the Intel486 DX CPU's PLOCK# protocol by providing a PLOCKEN configuration input. If PLOCKEN is detected high at RESET, the 82495DX CPLOCK# pin reflects the Intel486 DX CPU PLOCK# output for write cycles and the 82495DX follows certain PLOCK rules. The 82495DX's read cycles never generate CPLOCK#. If PLOCKEN is detected low at RESET, CPLOCK# is not generated and the PLOCK rules are not enforced.

CPLOCK# is driven like PLOCK# of the Intel486 DX CPU, but only for write cycles. It is active with CADS# of the first PLOCK write and indicates that the next write will be PLOCKed to it. To emulate the behavior of the Intel486 DX CPU's PLOCK# signal, MPLOCK# is asserted for write cycles as well as for multiple-dword read cycles. CPLOCK# goes inactive with CNA# or CRDY# of that first write cycle. The state machine for PLOCK is shown in Figure 5-12.

5.3.9 WARM RESET

The 82495DX allows the Intel486 DX CPU to be reset without resetting the cache subsystem, this is termed a warm-reset. This sequence is important for some PC software that uses the warm reset to return from protected mode to real mode of the CPU.

The 82495DX provides a special pin, called WRMRST, to help with this function. The reset input to the CPU (RESETCPU) is connected to the 82495DX WRMRST input within the CPU-Cache module (WRMRST does not appear on the CPU-Cache module's interface) so that the 82495DX knows when CPU is being reset.

To warm-reset the Intel486 DX CPU, the following must be done:

1. Assert HOLD to the CPU and wait for HLDA. This ensures that no CPU bus cycles are running that could corrupt the cache subsystem.
2. Assert RESET to the CPU only via RESETCPU.
3. Wait for CAHOLD to be inactive if it is not inactive already. This is so when RESETCPU is deasserted, the CPU does not go into self-test (note that although the Intel486 DX CPU may be resetting, the cache is allowed to be active. This is possible because the CPU allows back-invalidations during reset).
4. Deassert RESETCPU in the next clock. The 82495DX will not guarantee that CAHOLD will stay inactive beyond this.

During a warm reset sequence, the resets to the 82495DX and 82490DXs (RESET0 and RESET1) are not asserted. This sequence is implemented in state machine XWRMRST as shown Figure 5-13. The MBC signal WARM is used to generate RESETCPU as shown in the PLD equations in Section 7.

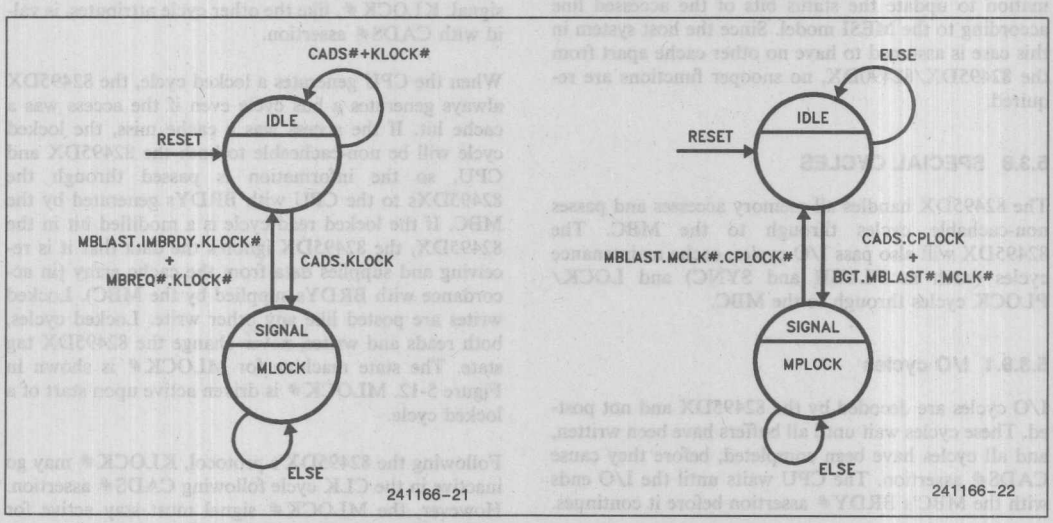


Figure 5-12. The MBC Implements the LOCK and PLOCK Protocol (XMLOCK and XMPLOCK)

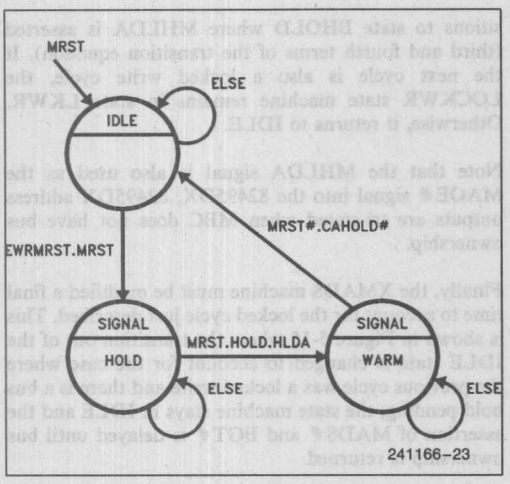


Figure 5-13. Generating a Warm Reset (XWRMRST)

In this MBC design, the host system Intel486 DX socket only includes a cold reset pin, MRST (since a separate warm reset is not necessary for the CPU by itself). Two options are available if software that uses the warm reset must be tried or evaluated. The first is to allow the system to start with a cold reset (jumper J9, EWRMRST# = 1) and then switch to warm resets before the software that uses them is started. The other option is to separately wire a warm reset signal directly from the host system.

5.3.10 MHOLD/MHLDA ARBITRATION

The bus arbitration state machine—XMHLDA—is actually a combination of two simple state machines as shown in Figure 5-14. For the most part, bus exchanges occur as a result of transitions between states IDLE and BHOLD of the first state machine shown. If a bus hold is being requested, the first state machine transitions to BHOLD where MHLDA is asserted to give away bus ownership if the MBC is not running locked or pseudo-locked cycles (first two terms in the transition equation).

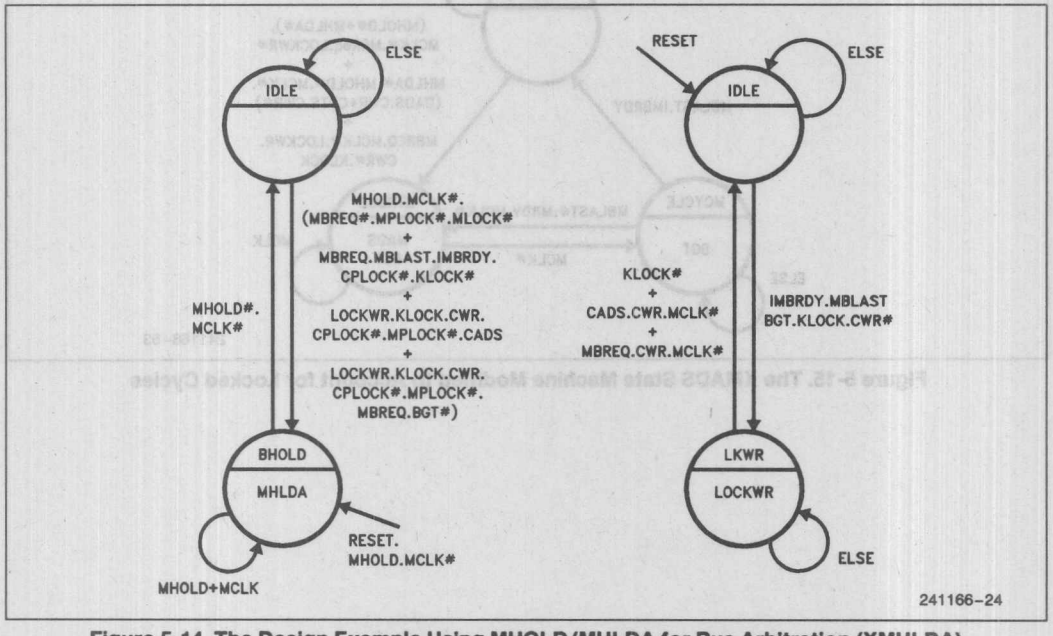


Figure 5-14. The Design Example Using MHOLD/MHLDA for Bus Arbitration (XMHLDA)

A second state machine is used to generate the signal LOCKWR. It helps dealing with multiple back-to-back atomic (locked) operations. An example of this is a continuous sequence of read-modify-write cycles. Under these conditions, the KLOCK# signal stays asserted for the entire sequence. This will increase the MHLDA latency since the transition from IDLE to BHOLD waits for the entire locked sequence to end. To minimize this latency, the boundary between one atomic operation and the next should be detected and the bus relinquished in between. This boundary is signaled by a locked write (end of atomic operation n) followed by a locked read (beginning of atomic operation $n+1$).

The LOCKWR state machine detects the occurrence of a locked write at which point LOCKWR is asserted. If the next cycle is a locked read and there is a bus MHOLD pending, then the first state machine transitions to state BHOLD where MHLDA is asserted (third and fourth terms of the transition equation). If the next cycle is also a locked write cycle, the LOCKWR state machine remains in state LKWR. Otherwise, it returns to IDLE.

Note that the MHLDA signal is also used as the MAOE# signal into the 82495DX; 82495DX address outputs are tri-stated when MBC does not have bus ownership.

Finally, the XMADS machine must be modified a final time to account for the locked cycle just described. This is shown in Figure 5-15 where the transition out of the IDLE state is changed to account for the case where the previous cycle was a locked write and there is a bus hold pending; the state machine stays in IDLE and the assertion of MADS# and BGT# is delayed until bus ownership is returned.

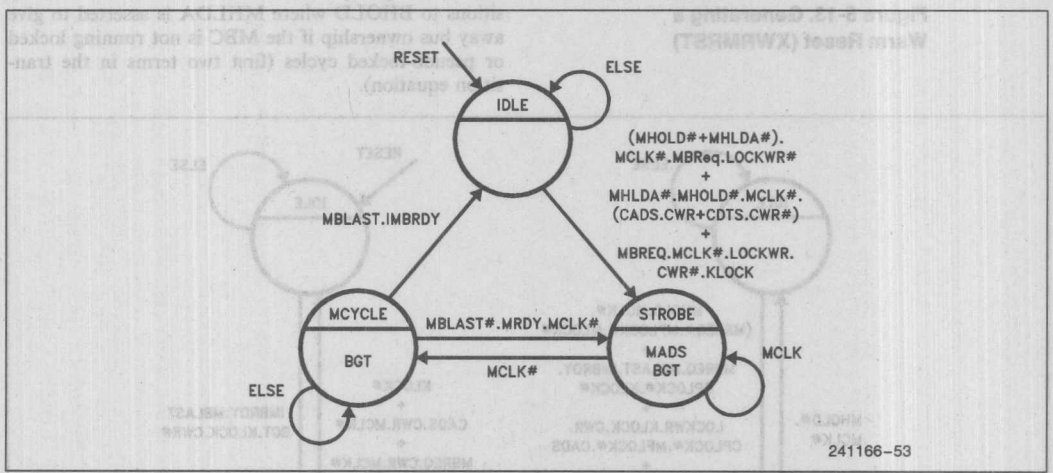


Figure 5-15. The XMADS State Machine Modified to Account for Locked Cycles

6.0 TIMING DIAGRAMS

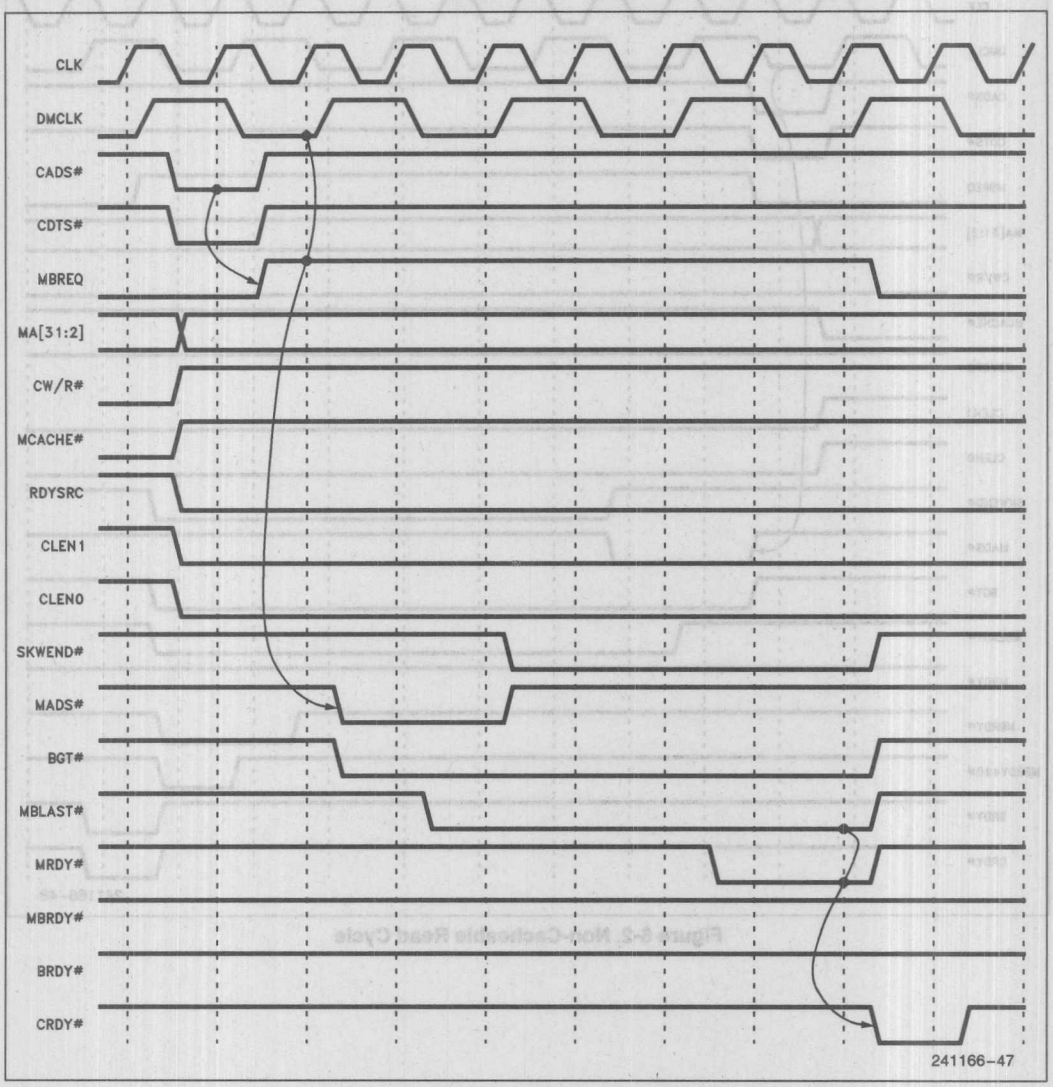


Figure 6-1. Non-Cacheable Write Cycle

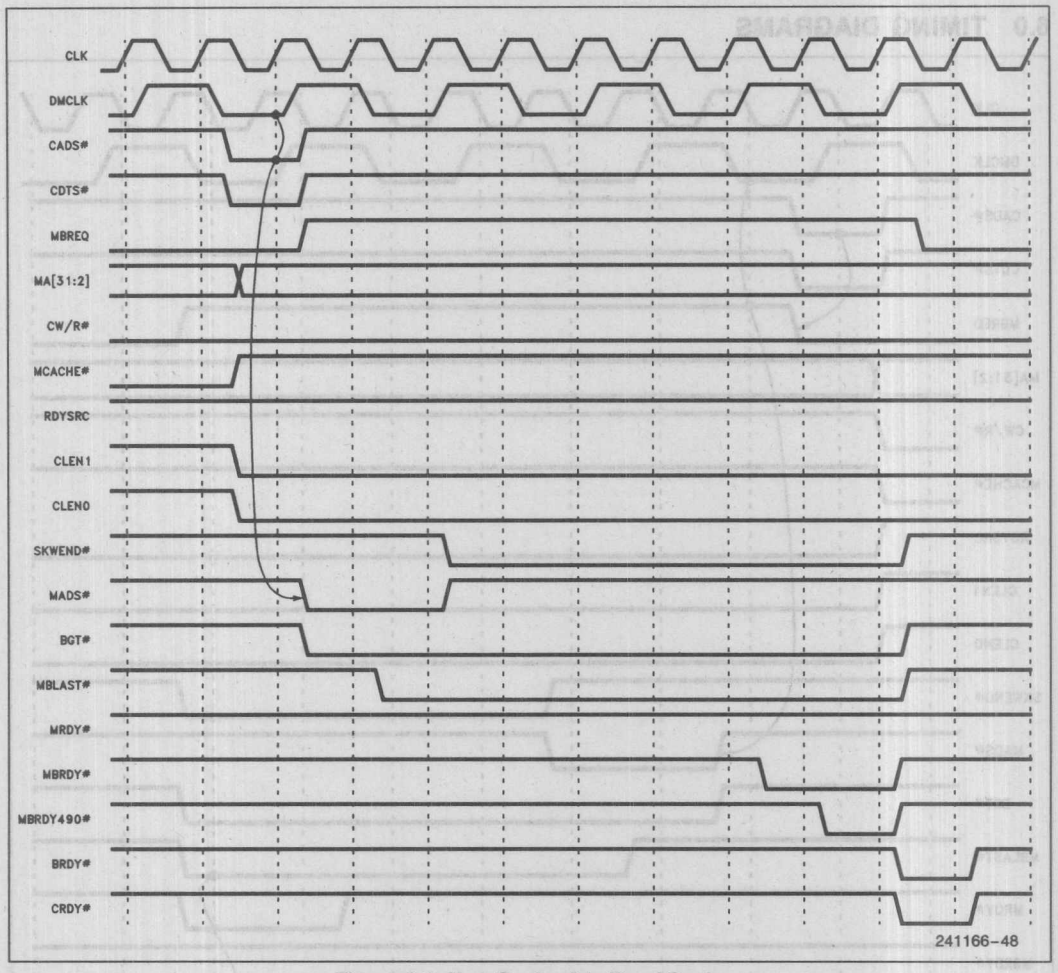


Figure 6-2. Non-Cacheable Read Cycle

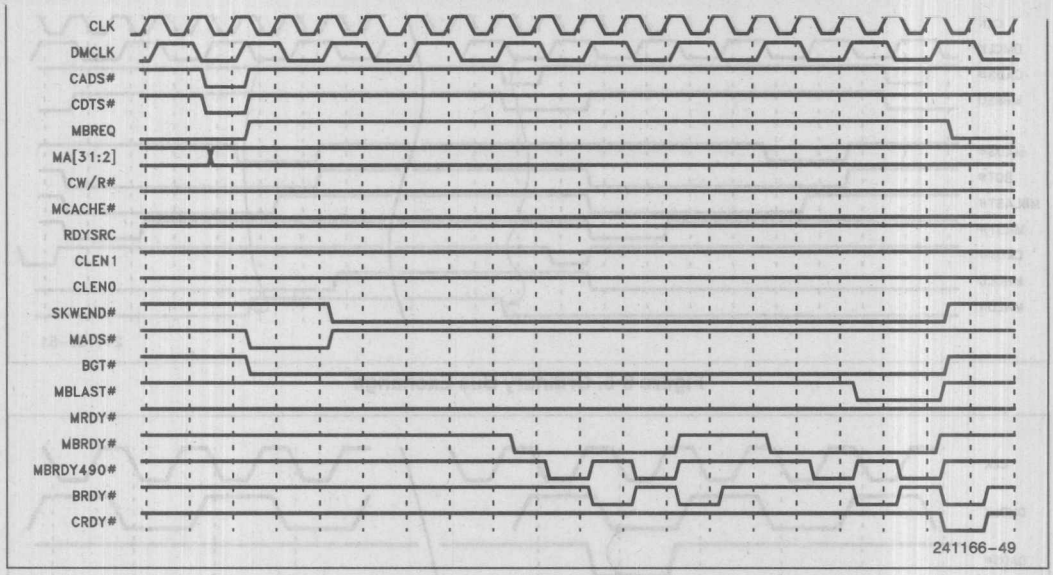


Figure 6-3. Line Fill

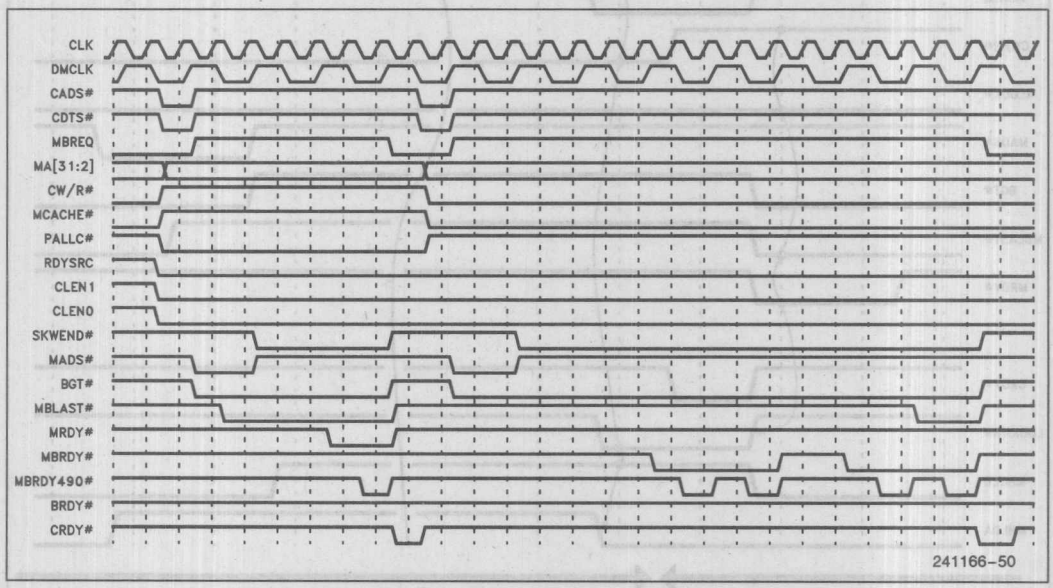


Figure 6-4. Write Miss followed by Allocation Cycle

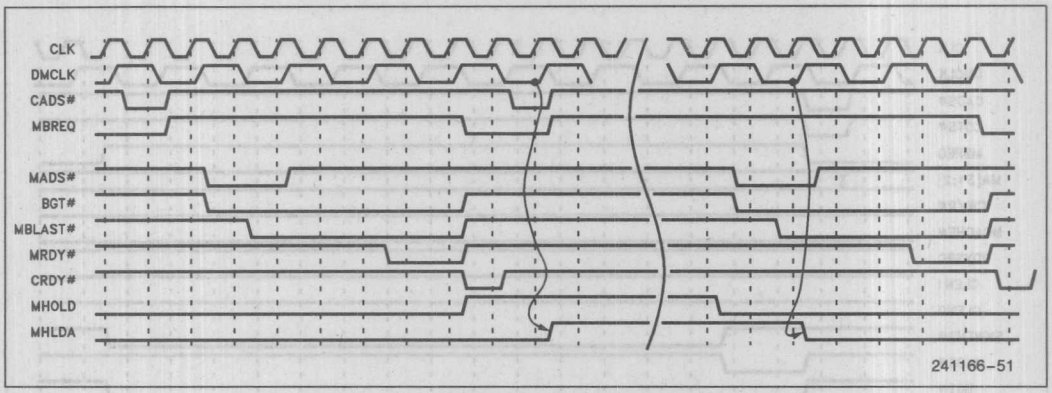


Figure 6-5. Ordinary Bus Exchange

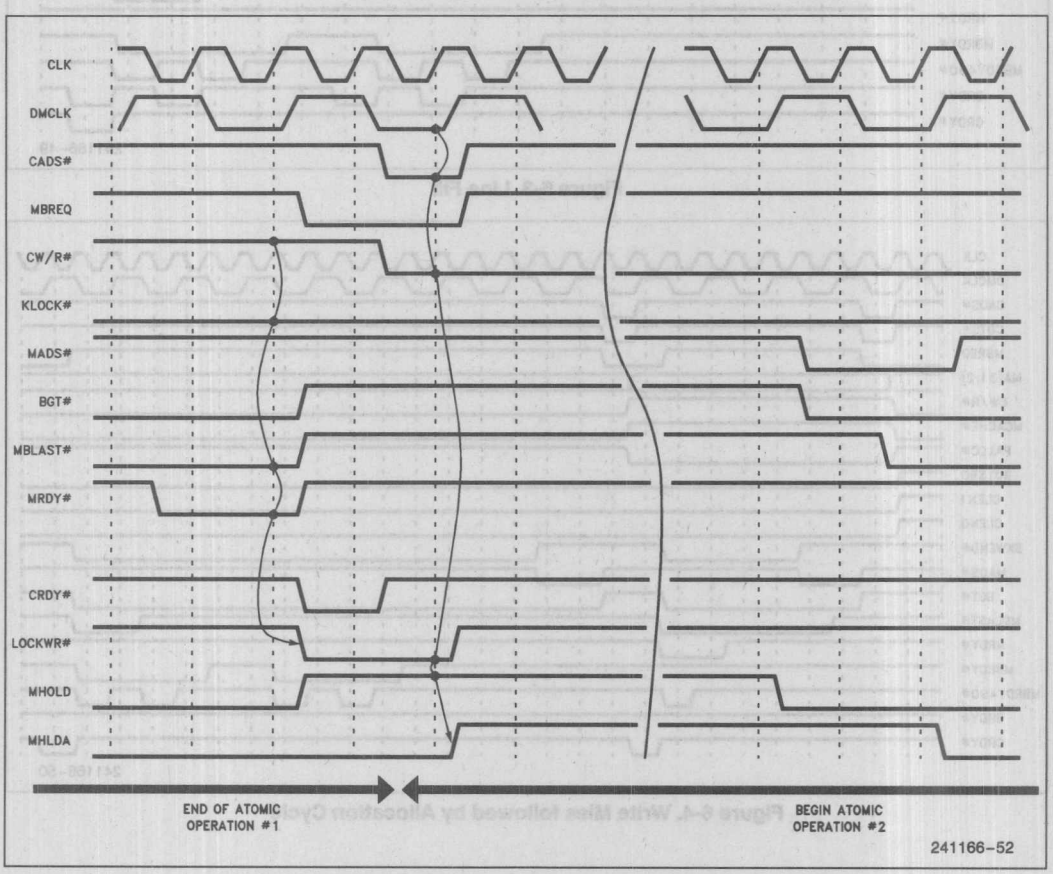


Figure 6-6. Bus Exchange between Two Atomic Operations

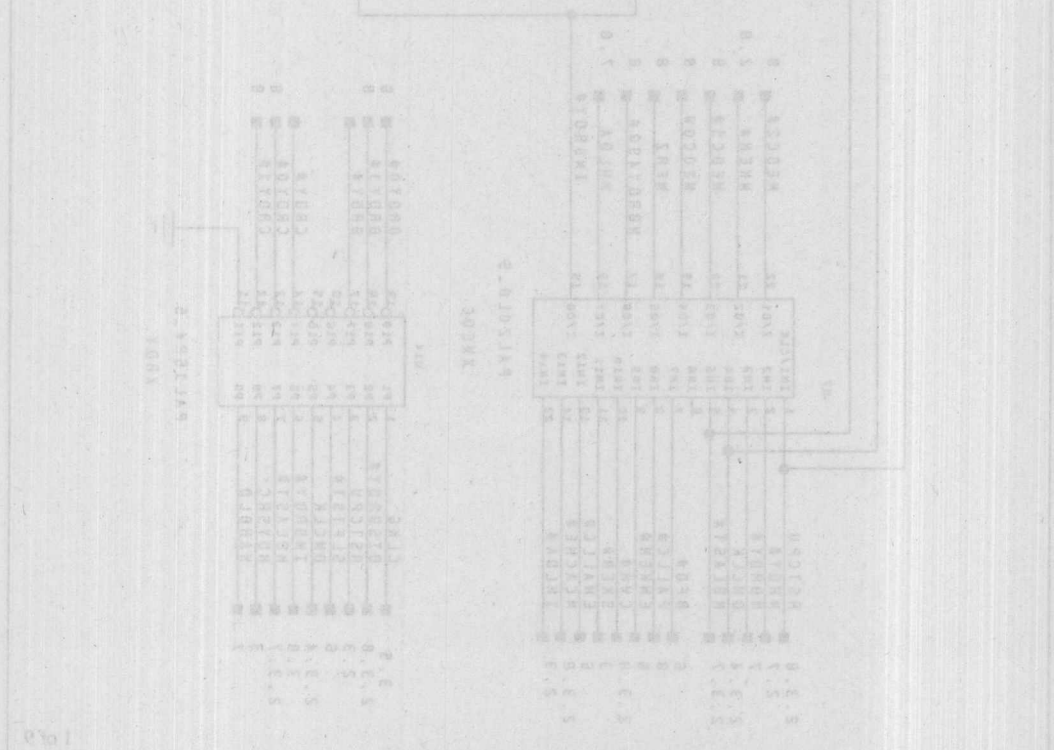
7.0 SCHEMATICS AND PLD CODES

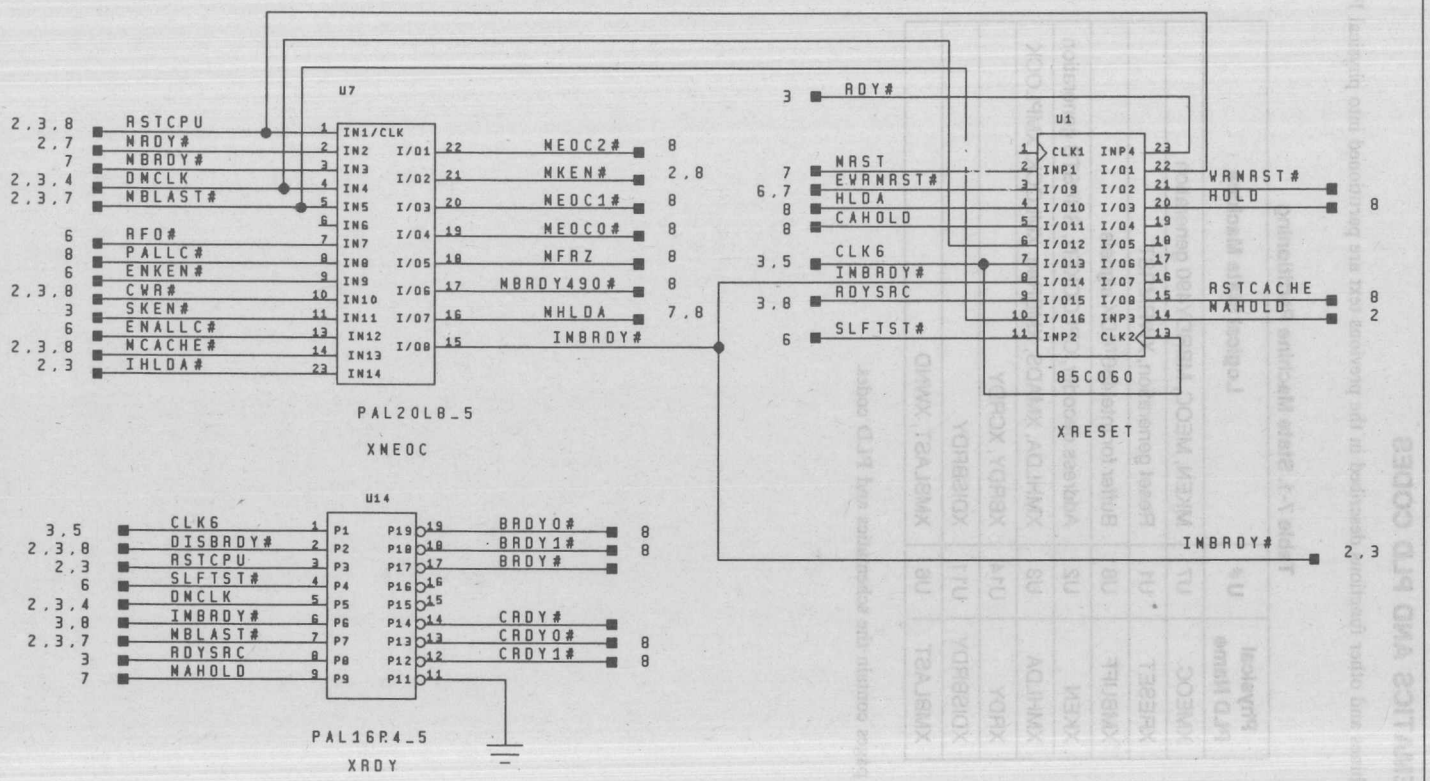
The state machines and other functions described in the previous text are partitioned into physical PLD devices as follows:

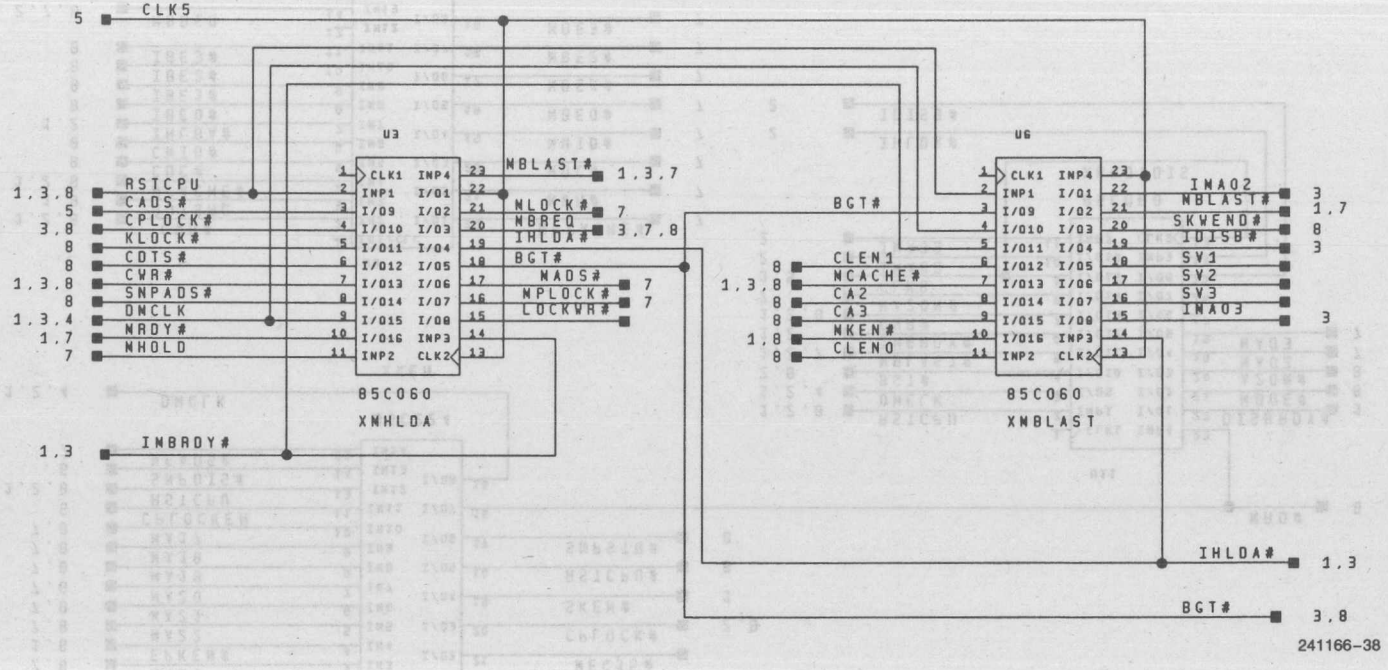
Table 7-1. State Machine Partitioning

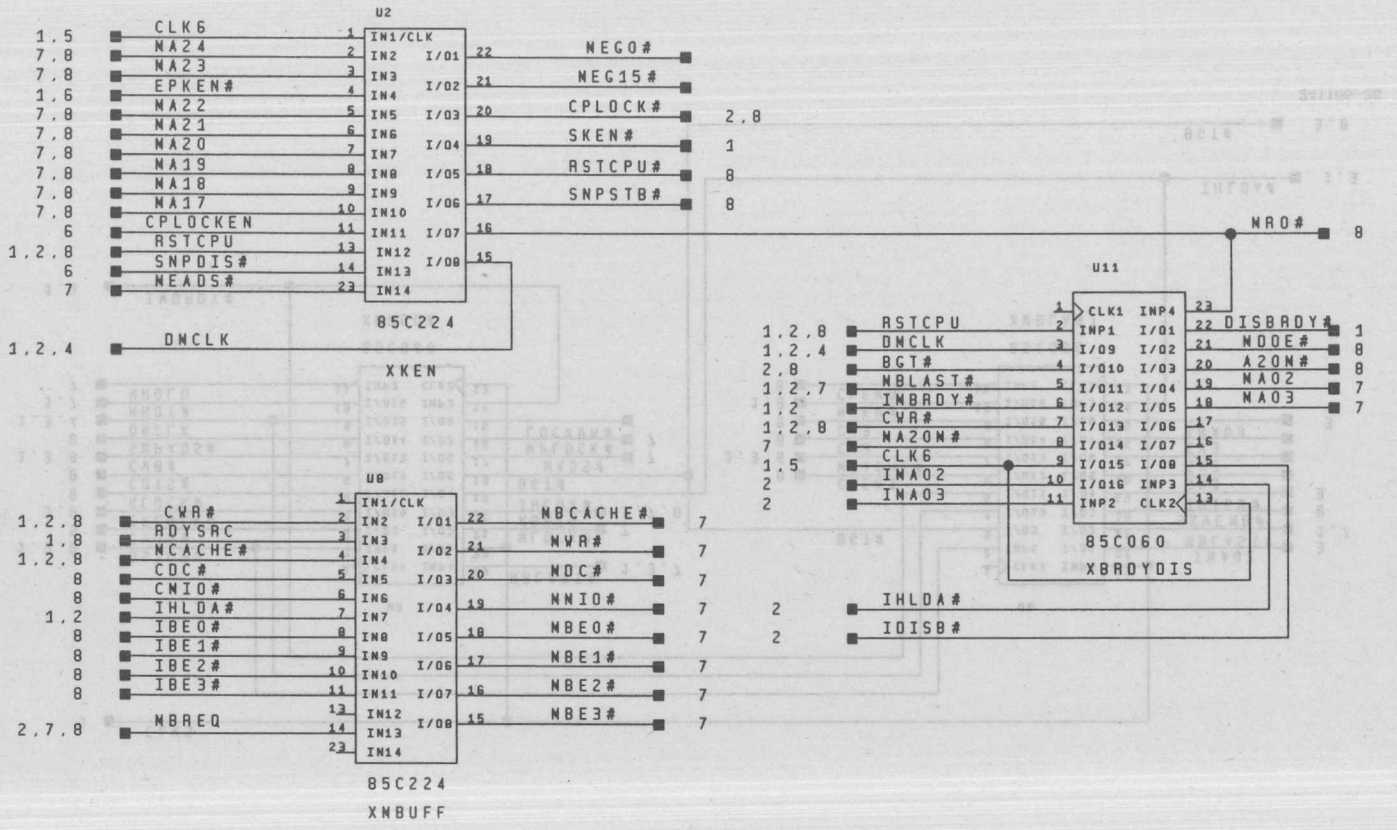
Physical PLD Name	U#	Logical State Machine
XMEOC	U7	MKEN, MEOC, MBRDY490 generation
XRESET	U1	Reset generation, XWRMRST
XMBUFF	U8	Buffer for Intel486™DX signals
XKEN	U2	Address decoder, CPLOCKEN, SNPSTB generation
XMHLDA	U3	XMHLDA, XMADS, XMBReq, XMLOCK, XMPLOCK
XRDY	U14	XBRDY, XCRDY
XDISBRDY	U11	XDISBRDY
XMBLAST	U6	XMBLAST, XWND

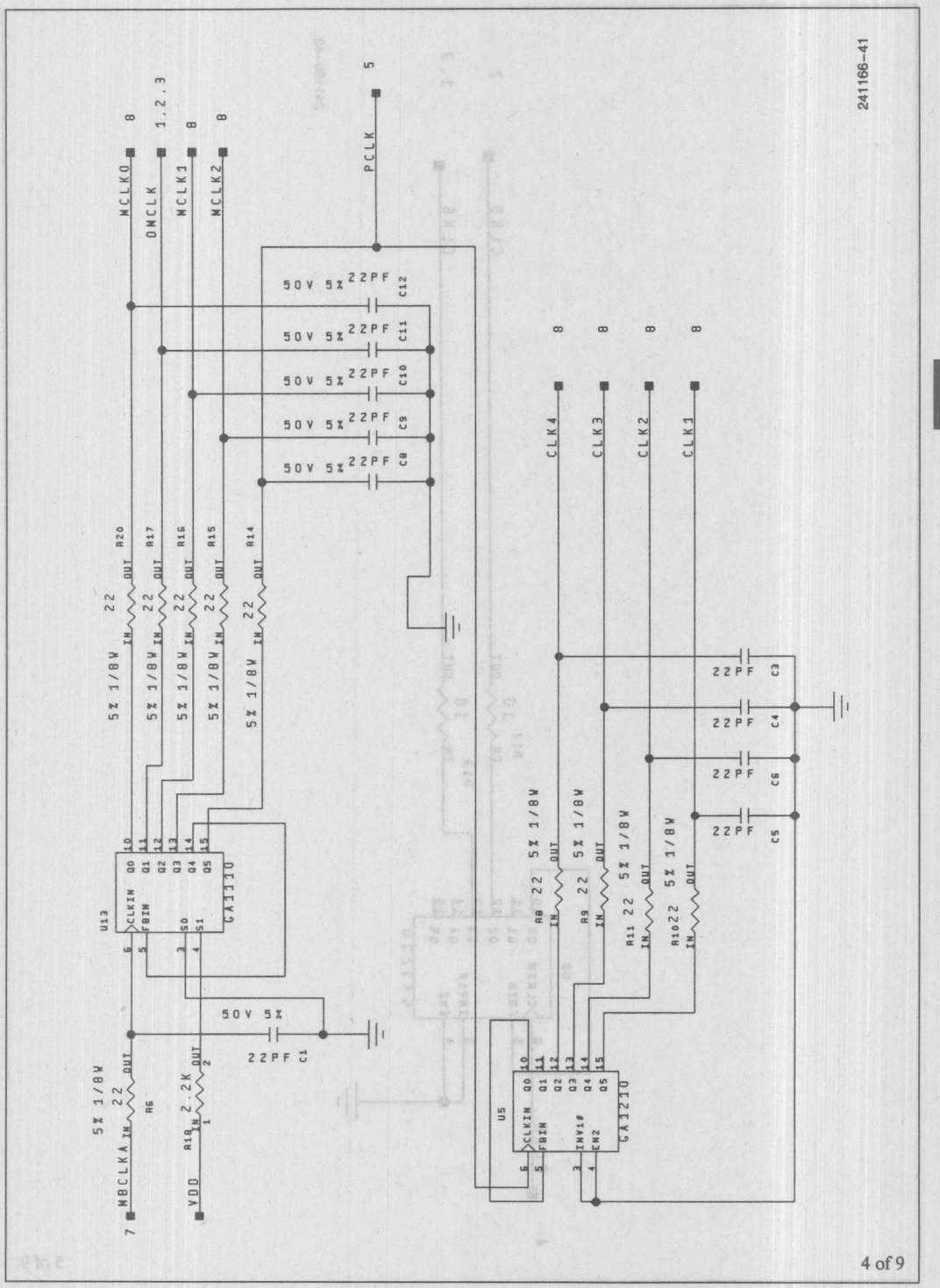
The following pages contain the schematics and PLD codes.

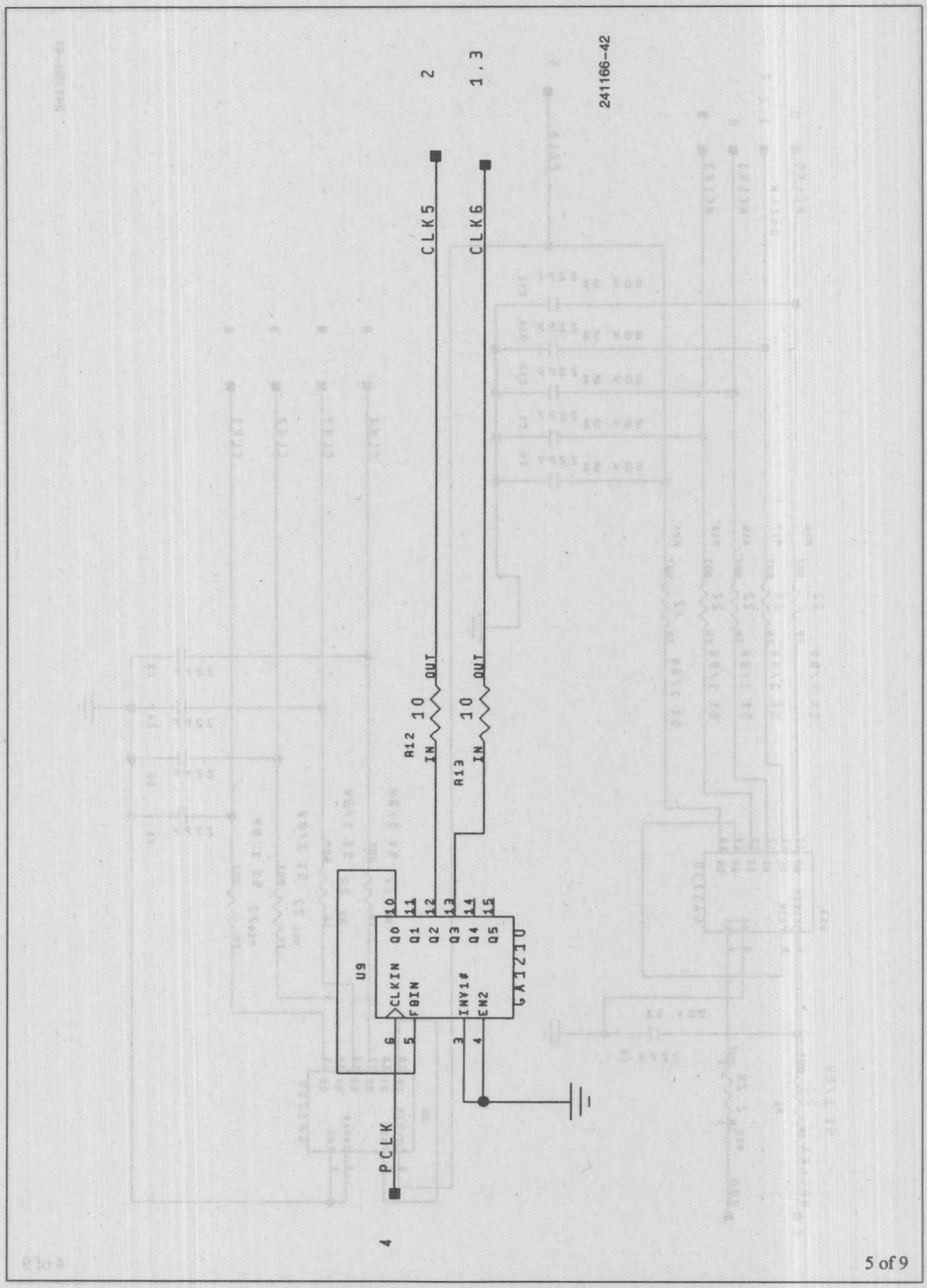


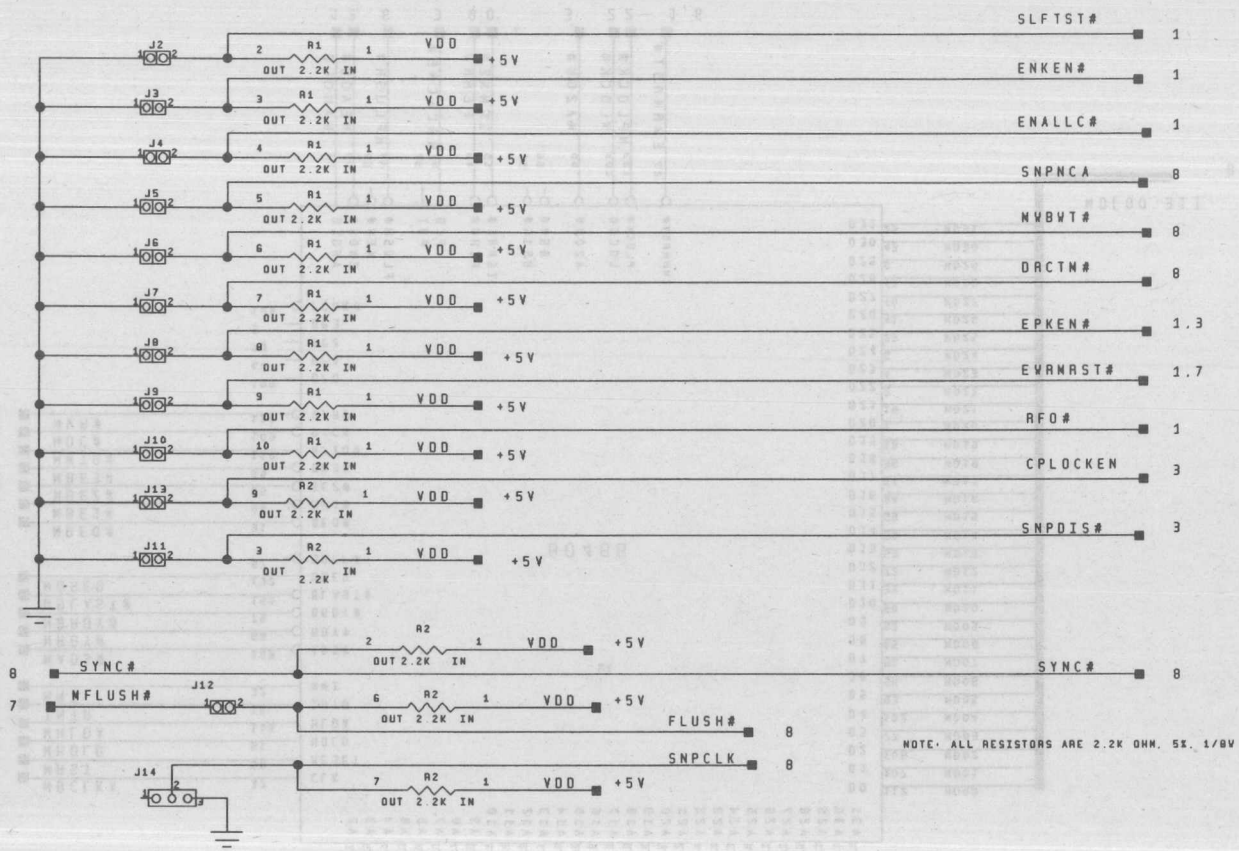

U7
2,3,8 RSTCPU
2,7 NRDY#
7 NBRDY#
2,3,4 DMCLK
2,3,7 NBLAST#
6 RFO#
8 PALLC#
6 ENKEN#
2,3,8 CWR#
3 SKEN#
6 ENALLC#
2,3,8 NCACHE#
2,3 IHLDA#
IN1/CLK
IN2
IN3
IN4
IN5
IN6
IN7
IN8
IN9
IN10
IN11
IN12
IN13
IN14
I/O1
I/O2
I/O3
I/O4
I/O5
I/O6
I/O7
I/O8
NEOC2# 8
NKEN# 2,8
NEOC1# 8
NEOC0# 8
NFRZ 8
NBRDY490# 8
NHLDA 7,8
INBRDY#
PAL20L8_5
XNEOC
3 RDY#
U1
7 NRST
6,7 EWRNRST#
8 HLDA
8 CAHOLD
3,5 CLK6
INBRDY#
3,8 RDYSRC
6 SLFTST#
CLK1
INP1
I/O9
I/O10
I/O11
I/O12
I/O13
I/O14
I/O15
I/O16
INP2
INP4
I/O1
I/O2
I/O3
I/O4
I/O5
I/O6
I/O7
I/O8
INP3
CLK2
WRNRST#
HOLD 8
RSTCACHE
NAHOLD 8 2
85C060
XRESET
INBRDY# 2,3
U14
3,5 CLK6
2,3,8 DISBRDY#
2,3 RSTCPU
6 SLFTST#
2,3,4 DMCLK
3,8 INBRDY#
2,3,7 NBLAST#
3 RDYSRC
7 NAHOLD
P1
P2
P3
P4
P5
P6
P7
P8
P9
P19
P18
P17
P16
P15
P14
P13
P12
P11
BRDY0# 8
BRDY1# 8
BRDY#
CRDY#
CRDY0# 8
CRDY1# 8
PAL16R4_5
XRDY
241166-37


5 CLK5
U3
1,3,8 RSTCPU
5 CADS#
3,8 CPLOCK#
8 KLOCK#
8 CDTS#
1,3,8 CWR#
8 SNPADS#
1,3,4 DNCLK
1,7 NRDY#
7 NHOLD
1,3 INBRDY#
CLK1 INP4
INP1 I/O1
I/O9 I/O2
I/O10 I/O3
I/O11 I/O4
I/O12 I/O5
I/O13 I/O6
I/O14 I/O7
I/O15 I/O8
I/O16 INP3
INP2 CLK2
NBLAST# 1,3,7
NLOCK# 7
NBREQ 3,7,8
IHLDA#
BGT#
NADS# 7
NPLOCK# 7
LOCKWR#
85C060
XNHLDA
U6
BGT#
8 CLEN1
1,3,8 NCACHE#
8 CA2
8 CA3
1,8 NKEN#
8 CLEN0
INA02
NBLAST# 3
SKWEND# 1,7
IOISB# 8
SV1 3
SV2
SV3
INA03 3
85C060
XNBLAST
IHLDA# 1,3
BGT# 3,8
241166-38


U2
CLK6
MA24
MA23
EPKEN#
MA22
MA21
MA20
MA19
MA18
MA17
CPLOCKEN
RSTCPU
SNPDIS#
MEADS#
IN1/CLK
IN2
IN3
IN4
IN5
IN6
IN7
IN8
IN9
IN10
IN11
IN12
IN13
IN14
I/O1
I/O2
I/O3
I/O4
I/O5
I/O6
I/O7
I/O8
MEG0#
MEG15#
CPLOCK#
SKEN#
RSTCPU#
SNPSTB#
85C224
DMCLK
XKEN
U8
CWR#
RDYSRC
MCACHE#
CDC#
CMIO#
IHLDA#
IBE0#
IBE1#
IBE2#
IBE3#
MBREQ
MBCACHE#
MWR#
MDC#
MMIO#
MBE0#
MBE1#
MBE2#
MBE3#
85C224
XMBUFF
U11
RSTCPU
DMCLK
BGT#
MBLAST#
IMBRDY#
CWR#
MA20M#
CLK6
IMA02
IMA03
CLK1
INP1
I/O9
I/O10
I/O11
I/O12
I/O13
I/O14
I/O15
I/O16
INP2
INP4
I/O1
I/O2
I/O3
I/O4
I/O5
I/O6
I/O7
I/O8
INP3
CLK2
MRO#
DISBRDY#
MDOE#
A20M#
MA02
MA03
IHLDA#
IDISB#
85C060
XBRDYDIS
241166-39


241166-41
4 of 9
MBCLKA
VDD
NCLKO
DNCLK
NCLK1
NCLK2
PCLK
CLK4
CLK3
CLK2
CLK1
U13
U5
GA1110
GA1210
5% 1/8W
50V 5%
22PF
2.2K

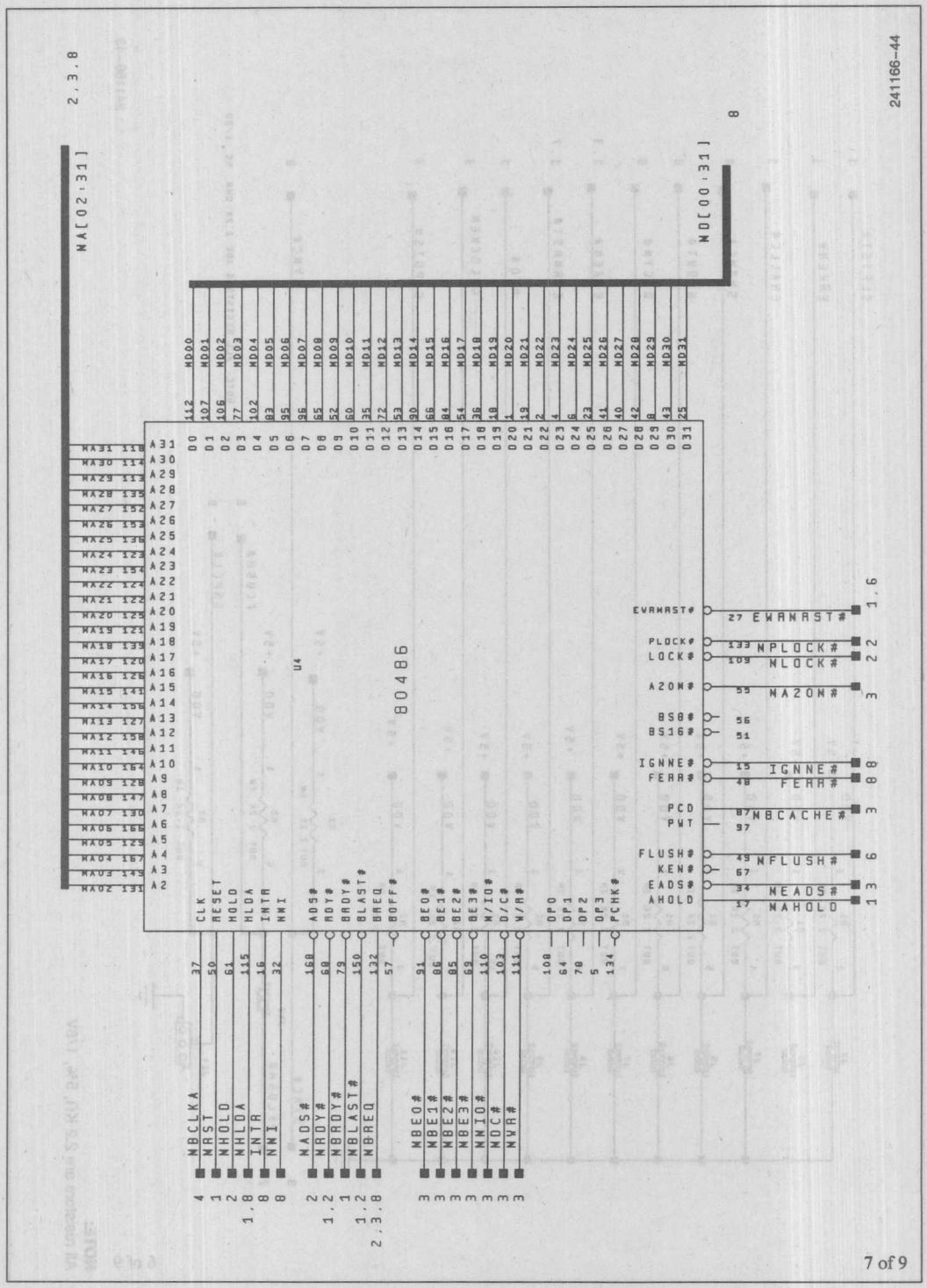

241166-42
PCLK
4
U9
CLKIN
FBIN
INV1#
EN2
Q0
Q1
Q2
Q3
Q4
Q5
CY7C210
R12
10
R13
10
IN
OUT
CLK5
CLK6
2
1,3

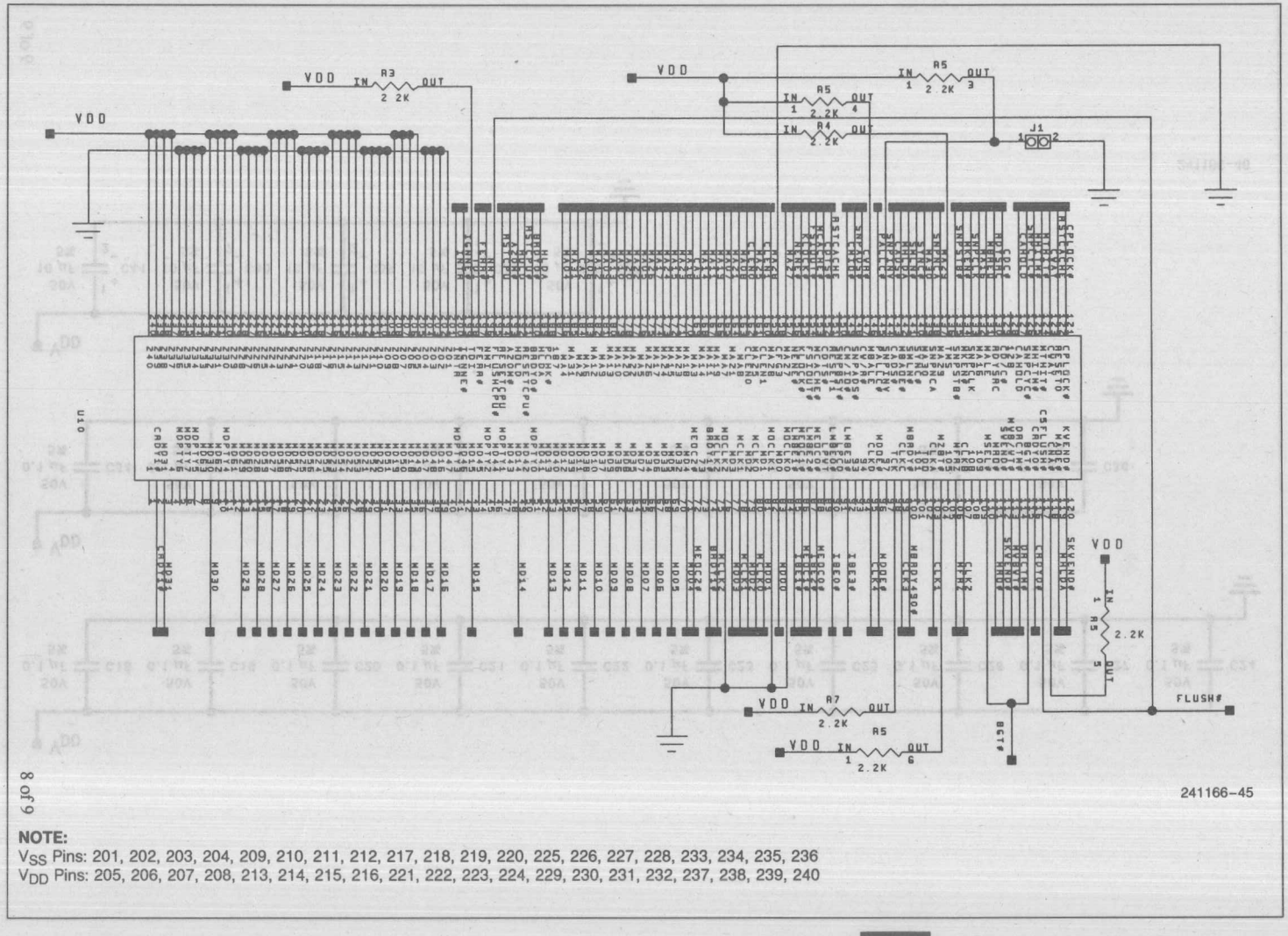


241166-43

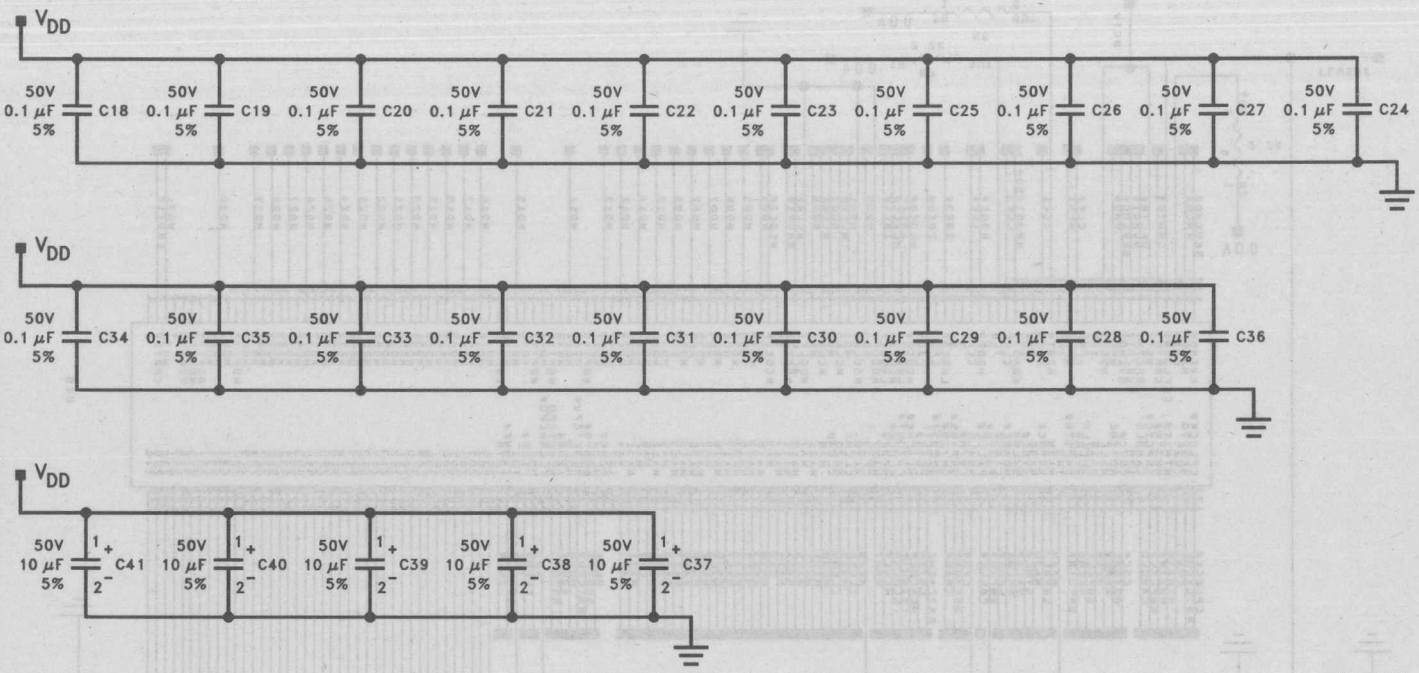
6 of 9

NOTE:
All resistors are 2.2 KΩ, 5%, 1/8V


241166-44
7 of 9
80486
U4
NA[02:31]
ND[00:31]
NBCLKA
NRST
NHOLD
NHLDA
INTR
NMI
NADS#
NRDY#
NBRDY#
NBLAST#
NBREQ
NBOFF#
NBE0#
NBE1#
NBE2#
NBE3#
NM/IO#
ND/C#
NW/R#
DP0
DP1
DP2
DP3
PCHK#
EWRMRST#
PLOCK#
LOCK#
A20M#
BS8#
BS16#
IGNNE#
FERR#
PCD
PWT
FLUSH#
KEN#
EADS#
AHOLD
EWRMRST
NPLOCK#
NLOCK#
NA20M#
IGNNE#
FERR#
NWBCACHE#
NFLUSH#
NEADS#
NAHOLD



NOTE:
V_{SS} Pins: 201, 202, 203, 204, 209, 210, 211, 212, 217, 218, 219, 220, 225, 226, 227, 228, 233, 234, 235, 236
V_{DD} Pins: 205, 206, 207, 208, 213, 214, 215, 216, 221, 222, 223, 224, 229, 230, 231, 232, 237, 238, 239, 240



241166-46

intel®

```
TITLE            XMEOC
PATTERN          F7A.PDS
REVISION         A
AUTHOR           MODULES
COMPANY          INTEL CORP.
DATE             8/17/91

CHIP    XMEOC    PAL20L8

RSTCPU /MRDY /MBRDY MCLK /MBLAST NC1 /RFO /PALLC /ENKEN /CWR /SKEN GND
/ENALLC /MCACHE /IMBRDY MHLDA /MBRDY490 /MFRZ /MEOC0 /MEOC1 /MKEN /MEOC2
/IHLDA VCC


EQUATIONS

MEOC0   = MRDY * /MCLK * MBLAST
        + MBRDY * /MCLK * MBLAST

MEOC1   = MRDY * /MCLK * MBLAST
        + MBRDY * /MCLK * MBLAST

MEOC2   = MRDY * /MCLK * MBLAST
        + MBRDY * /MCLK * MBLAST

IMBRDY  = MRDY * /MCLK
        + MBRDY * /MCLK

MFRZ    = /RSTCPU * MRDY * /MCLK * MBLAST * MKEN * PALLC * RFO
        + /RSTCPU * MBRDY * /MCLK * MBLAST * MKEN * PALLC * RFO

MBRDY490 = MRDY * /MCLK
         + MBRDY * /MCLK

MKEN    = ENKEN * CWR * SKEN
        + ENKEN * /CWR * ENALLC * PALLC * SKEN
        + /CWR * MCACHE

MHLDA   = IHLDA

;---------------------------------------------------------------------
;
;MKEN is generated in this PLD.  ENKEN is a configurable option
;for enabling/diabling cache.  ENALLC allows allocation cycles
;to take place in the case of write-miss cycles. These can be
;read-for-ownership cycles using the RFO jumper. This freezes
;the memory write data in 82490 during the write-miss cycle
;and is combined  with the rest of the data fetched during
;the allocation.
;
;---------------------------------------------------------------------
```

241166-25

2

```
TITLE           XRESET
PATTERN         F1A.PDS
REVISION        A
AUTHOR          MODULES
COMPANY         INTEL CORP.
DATE            8/16/91

OPTIONS TURBO = ON

CHIP    XRESET  85C060

CLK MRST /EWRMRST HLDA CAHOLD MCLK ACLK /IMBRDY RDYSRC /MBLAST /SLFTST GND
/OE MAHOLD RSTCACHE /CRDY1 /BRDY0 /CRDY0 /BRDY1 HOLD /WARM  RSTCPU /DISB
VCC

EQUATIONS


HOLD    := EWRMRST * MRST * /WARM
         + EWRMRST * HOLD * /HLDA

HOLD.CLKF = ACLK

WARM    :=  MRST * HOLD * HLDA * EWRMRST
          + MRST * WARM  * EWRMRST
          + CAHOLD * WARM  * EWRMRST

WARM.CLKF = ACLK

RSTCPU  = MRST * /MCLK * /EWRMRST
        + RSTCPU * MRST * /EWRMRST
        + RSTCPU * MCLK * /EWRMRST
        + WARM  * EWRMRST

RSTCACHE = MRST * /MCLK * /EWRMRST
         + RSTCACHE * MRST * /EWRMRST
         + RSTCACHE * MCLK * /EWRMRST

;---------------------------------------------------------------

;This state machine generates reset lines to CPU,82495,82490.
;Because the i486 motherboard socket has only one RESET pin
;it can be either used to reset all of them or only the CPU.
;A configuration jumper /EWRMRST selects between WARMRESET and
;full-reset. If the system has to be reset the jumper has to be
;removed so that 82495DX also gets reset and cache is flushed.
;Or /EWRMRST can be connected from a software configurable port
;pin to the i486 socket.
;
;---------------------------------------------------------------
```

241166-26

```
TITLE          XMBUFF
PATTERN         F8A.PDS
REVISION        A
AUTHOR          MODULES
COMPANY         INTEL CORP.
DATE            6/3/91

OPTIONS TURBO = ON

CHIP    XMBUFF  85C224

NC4 /CWR RDYSRC /MCACHE /CDC /CMIO /IHLDA /IBE0 /IBE1 /IBE2 /IBE3 GND
NC1 MBREQ /MBE3 /MBE2 /MBE1 /MBE0 /MMIO /MDC /MWR /MBCACHE NC3 VCC

EQUATIONS

MBCACHE      = MCACHE

MBCACHE.TRST = /IHLDA

MWR          = CWR

MWR.TRST     = /IHLDA

MDC          = CDC

MDC.TRST     = /IHLDA

MMIO         = CMIO * MBREQ

MMIO.TRST    = /IHLDA

MBE0.TRST    = /IHLDA

MBE0         = IBE0 + (MCACHE * /RDYSRC)

MBE1.TRST    = /IHLDA

MBE1         = IBE1 + (MCACHE * /RDYSRC)

MBE2.TRST    = /IHLDA

MBE2         = IBE2 + (MCACHE * /RDYSRC)

MBE3.TRST    = /IHLDA

MBE3         = IBE3 + (MCACHE * /RDYSRC)

;---------------------------------------------------------------
;
;Generates the memory bus control signals and byte enables.
;Note that all byte enables are activated when RDYSRC is inactive
;and MCACHE is active.  This is for write-back cycles.
;
;---------------------------------------------------------------
```

241166-27

```
TITLE           XKEN
PATTERN         F2A.PDS
REVISION        A
AUTHOR          MODULES
COMPANY         INTEL CORP.
DATE            7/22/91

OPTIONS TURBO = ON

CHIP    XKEN    85C224


CLK MA24 MA23 /EPKEN MA22 MA21 MA20 MA19 MA18 MA17 CPLOCKEN GND
RSTCPU /SNPDIS MCLK /MRO /SNPSTB /RESETCPU /SKEN /CPLOCK /MEG15 /MEG0 /MEADS VCC

EQUATIONS

MEG0  = /MA24 * /MA23 * /MA22 * /MA21 * /MA20

MEG15 = MA24 * MA23 * MA22 * MA21 * MA20

SKEN  = /MEG0 * /MEG15
      + MEG0 * /MA19
      + MEG0 * MA19 * /MA18 * /MA17
      + MEG15 * /MA19
      + EPKEN * MEG0 * MA19 * MA18 * MA17

MRO   = EPKEN * MEG0 * MA19 * MA18 * MA17

CPLOCK.TRST = /CPLOCKEN * RSTCPU

CPLOCK = RSTCPU

RESETCPU = RSTCPU

SNPSTB = MEADS * /MCLK * /SNPDIS

;-------------------------------------------------------------
;CPLOCK/ driven by Debug board at reset time as PLOCKEN pin.
;After reset the pin is floated by Debug board since CPLOCK/
;is output signal from 82495.

;MRO signal is generated for the last 128KB of 1MB (system BIOS
;area). It is a configurable option (EPKEN/). System KEN generated
;based on the address decoding. Cache is disabled only from
;640KB-1MB and between 15.5MB to 16MB. BIOS area can be enabled
;by making EPKEN active.

;-------------------------------------------------------------
```

241166-28

```
TITLE           XMHLDA
PATTERN         F3A.PDS
REVISION        A
AUTHOR          MODULES
COMPANY         INTEL CORP.
DATE            9/03/91

OPTIONS         TURBO = ON

CHIP    XMHLDA  85C060

CLK1 RSTCPU /CADS /CPLOCK /KLOCK /CDTS /CWR /SNPADS MCLK /MRDY MHOLD  GND
CLK2 /IMBRDY /LOCKWR /MPLOCK /MADS /BGT /IHLDA MBREQ /MLOCK ACLK /MBLAST VCC

EQUATIONS

IHLDA       := RSTCPU * MHOLD * /MCLK
             + /RSTCPU * MHOLD * IHLDA
             + /RSTCPU * MCLK * IHLDA
             + /MLOCK * /MPLOCK * MHOLD * /MCLK * /MBREQ
             + /KLOCK * /CPLOCK * MHOLD * IMBRDY * MBLAST * /MCLK * MBREQ
             + /RSTCPU * LOCKWR * CADS * CWR * KLOCK * MHOLD * /MCLK *
             /CPLOCK * /MPLOCK * /MBREQ
             + /RSTCPU * LOCKWR * MBREQ * CWR * KLOCK * MHOLD * /MCLK *
             /CPLOCK * /MPLOCK * /BGT

IHLDA.CLKF   = ACLK

MBREQ       := /RSTCPU * /SNPADS * /IMBRDY * MBREQ
             + /RSTCPU * /SNPADS * /MBLAST * MBREQ
             + /RSTCPU * CDTS * /CWR * /MBREQ
             + /RSTCPU * CADS * CWR * /MBREQ

MBREQ.CLKF   = ACLK

MADS        := /RSTCPU * MCLK * MADS
             + /RSTCPU * /IHLDA * MBREQ * /MCLK * /BGT * /LOCKWR
             + /RSTCPU * /MHOLD * MBREQ * /MCLK * /BGT * /LOCKWR
             + /RSTCPU * MRDY * /MBLAST * /MCLK * /MADS * BGT
             + /RSTCPU * /IHLDA * /MHOLD * CDTS * /CWR * /MCLK * /BGT
             + /RSTCPU * /IHLDA * /MHOLD * CADS * CWR * /MCLK * /BGT
             + /RSTCPU * MBREQ * /MCLK * /BGT * LOCKWR * /CWR * KLOCK

MADS.CLKF    = ACLK

BGT         := /RSTCPU * /IHLDA * MBREQ * /MCLK * /BGT * /LOCKWR
             + /RSTCPU * /MHOLD * MBREQ * /MCLK * /BGT * /LOCKWR
             + /RSTCPU * /IHLDA * /MHOLD * CDTS * /CWR * /MCLK * /BGT
             + /RSTCPU * /IHLDA * /MHOLD * CADS * CWR * /MCLK * /BGT
             + /RSTCPU * MADS
             + /RSTCPU * /IMBRDY * BGT
             + /RSTCPU * /MBLAST * BGT
             + /RSTCPU * MBREQ * /MCLK * /BGT * LOCKWR * /CWR * KLOCK

BGT.CLKF     = CLK2

MLOCK       := /RSTCPU * CADS * KLOCK
             + /RSTCPU * KLOCK * MLOCK
             + /RSTCPU * /MBLAST * MLOCK * MBREQ
             + /RSTCPU * /IMBRDY * MLOCK * MBREQ

MLOCK.CLKF   = ACLK

MPLOCK      := /RSTCPU * CADS * CPLOCK
             + /RSTCPU * CPLOCK * MPLOCK
             + /RSTCPU * /MBLAST * MPLOCK
             + /RSTCPU * MCLK * MPLOCK
             + /RSTCPU * BGT * /MBLAST * /MCLK

MPLOCK.CLKF  = ACLK

LOCKWR      := /RSTCPU * BGT * KLOCK * /CWR * IMBRDY * MBLAST
             + /RSTCPU * LOCKWR * KLOCK * /CADS * /MBREQ
             + /RSTCPU * LOCKWR * KLOCK * /CWR
             + /RSTCPU * LOCKWR * KLOCK * MCLK

LOCKWR.CLKF  = ACLK

;---------------------------------------------------------------
;MADS is generated in this state machine.It is activated as a result
;of CADS#(read cycle) or CDTS# (write cycle).For cycles which the
```

241166-29

```
;memory system does not want to burst and for write-backs the state
;machine generates MADS# after each MRDY# until the last transaction
;on the memory bus.
;
;MBREQ keeps track of the pending request from the Module and used
;to generate MADS# after IHLDA becomes inactive.
;
;IHLDA is generated when MHOLD is sampled active and the bus is not
;locked by KLOCK# or PLOCK#.But two atomic cycles occur back-to-back
;then IHLDA is generated before the start of the second atomic cycle.
;LOCKWR# is used to keep track of the write-portion of the atomic
;cycle and if LOCKWR# is active and the next cycle is a read then
;IHLDA is generated if MHOLD is pending at that time.
;
;Memory bus MLOCK and MPLOCK  are also generated in this state machine.
;Since the i486 LOCK remains active until after the last ready this
;state machine extends 82495 KLOCK until after the last ready from
;memory.
;
;-------------------------------------------------------------------
```

241166-30

```
TITLE           XRDY
PATTERN         F14A.PDS
REVISION        A
AUTHOR          MODULES
COMPANY         INTEL CORP.
DATE            7/30/91

CHIP    XRDY  PAL16R4

CLK /DISB RSTCPU /SLFTST MCLK /IMBRDY /MBLAST RDYSRC MAHOLD GND
/OE /CRDY1 /CRDY0 /CRDY NC1 NC2 /BRDY /BRDY1 /BRDY0 VCC

EQUATIONS


BRDY    := /RSTCPU * IMBRDY * RDYSRC * /DISB
         + /RSTCPU * IMBRDY * BRDY * /DISB

CRDY    := /RSTCPU * MBLAST * IMBRDY * /CRDY
         + RSTCPU * SLFTST * MAHOLD

BRDY0    = BRDY

BRDY1    = BRDY

CRDY0    = CRDY

CRDY1    = CRDY

;-------------------------------------------------------------------
;
;This state machine generates BRDY to the CPU ,82495 and 82490.
;BRDY is generated after sampling an active IMBRDY (MRDY or MBRDY).
;CRDY is generated after sampling active IMBRDY and MBLAST.
;
;-------------------------------------------------------------------
```

241166-31

```
TITLE           XBRDYDIS
PATTERN         F11A.PDS
REVISION        A
AUTHOR          MODULES
COMPANY         INTEL CORP.
DATE            8/15/91

OPTIONS TURBO = ON

CHIP    XKEN    85C060

NC1 RSTCPU MCLK /BGT /MBLAST /IMBRDY /CWR /MA20M ACLK IMA02 IMA03 GND
CLK2 /IHLDA /IDISB NC5 NC6 MA03 MA02 /A20M /MDOE /DISB /MRO VCC

EQUATIONS

DISB    := /RSTCPU * BGT * IDISB * MRO
         + /RSTCPU * BGT * DISB * /IMBRDY
         + /RSTCPU * BGT * DISB * /MBLAST

DISB.CLKF = ACLK

MDOE     = BGT * /CWR

A20M     = MA20M * /MCLK
         + A20M * MCLK
         + A20M * MA20M

MA02     = IMA02

MA02.TRST  = /IHLDA

MA03     = IMA03

MA03.TRST  = /IHLDA

;-------------------------------------------------------------
;
;This PLD provides tri-state control for burst address lines
;MA2,MA3 and also generates MDOE used to enable 82490 data
;buffers during write-cycles.
;DISB is used in the XRDY  state machine to control the number
;of BRDYs to CPU during MRO cycles.
;
```

241166-32

2

```
TITLE           XMBLAST
PATTERN         F6A.PDS
REVISION        A
AUTHOR          MODULES
COMPANY         INTEL CORP.
DATE            7/17/91

OPTIONS TURBO = ON

CHIP    XMBLAST 85C060

CLK RSTCPU /BGT MCLK /IMBRDY CLEN1 /MCACHE CA2 CA3 /MKEN CLEN0 GND
NC1 /IHLDA MA3 /SV3 /SV2 /SV1 /IDISB /SKWEND /MBLAST MA2 ACLK VCC


EQUATIONS

SV3     := /RSTCPU * /IMBRDY * SV3
         + /RSTCPU * IMBRDY * /SV3 * SV2
         + /RSTCPU * IMBRDY * /SV3 * SV1
         + /RSTCPU * /CLEN1 * /CLEN0 * /IMBRDY * SKWEND * /MKEN * /SV2 * SV1
         + /RSTCPU * /MCACHE * /CLEN1 * /CLEN0 * BGT * /SV3 * /SV2 * /SV1

SV3.CLKF = ACLK

SV2     := /RSTCPU * /IMBRDY * SV2
         + /RSTCPU * /SV3 * SV2
         + /RSTCPU * IMBRDY * SV3 * /SV2 * SV1
         + /RSTCPU * /MCACHE * /CLEN1 * CLEN0 * BGT * /SV3 * /SV1
         + /RSTCPU * /CLEN1 * CLEN0 * /IMBRDY * SKWEND * /MKEN * /SV3 * SV1

SV2.CLKF = ACLK

SV1     := /RSTCPU * CLEN1 * /IMBRDY * SV1
         + /RSTCPU * /IMBRDY * /SKWEND * SV1
         + /RSTCPU * /IMBRDY * MKEN * SV1
         + /RSTCPU * /IMBRDY * SV3 * SV1
         + /RSTCPU * /SV3 * SV2 * SV1
         + /RSTCPU * IMBRDY * /SV2 * SV1
         + /RSTCPU * CLEN1 * BGT * /SV3 * /SV2
         + /RSTCPU * MCACHE * BGT * /SV3 * /SV2 * /SV1

SV1.CLKF = ACLK

 MBLAST := /RSTCPU * IMBRDY * /SV3 * SV2
         + /RSTCPU * /MCACHE * /CLEN1 * /CLEN0 * BGT * /SV3 * /SV2 * /SV1
         + /RSTCPU * /IMBRDY * SV3 * SV2
         + /RSTCPU * /IMBRDY * SV3 * /SV1
         + /RSTCPU * /CLEN1 * /CLEN0 * /IMBRDY * SKWEND * /MKEN * /SV3 * /SV2
         * SV1

MBLAST.CLKF = ACLK

 /MA2    := RSTCPU * /CA2
         + /CA2 * IMBRDY * SV3
         + /CA2 * /IMBRDY * /SV3
         + /CA2 * /SV2 * /SV1
         + /RSTCPU * CA2 * /IMBRDY * SV3 * SV2
         + /RSTCPU * CA2 * IMBRDY * /SV3 * SV2
         + /RSTCPU * CA2 * /IMBRDY * SV3 * SV1
         + /RSTCPU * CA2 * IMBRDY * /SV3 * SV1

/MA2.CLKF   = ACLK

  /MA3  := RSTCPU * /CA3
         + /CA3 * /SV1
         + /CA3 * /IMBRDY * /SV2
         + /CA3 * /SV3 * /SV2
         + /CA3 * IMBRDY * SV3 * SV2
         + /RSTCPU * CA3 * /IMBRDY * SV2 * SV1
         + /RSTCPU * CA3 * /SV3 * SV2 * SV1
         + /RSTCPU * CA3 * IMBRDY * SV3 * /SV2 * SV1

/MA2.CLKF   = ACLK

SKWEND := RSTCPU
        + /RSTCPU * BGT * /MCLK * /SKWEND
        + /RSTCPU * BGT * SKWEND * /IMBRDY
```

241166-33

```
        + /RSTCPU * BGT * SKWEND * /MBLAST

SKWEND.CLKF   = ACLK

IDISB      := /RSTCPU * /IDISB * MKEN * MCACHE * /CLEN1 * /CLEN0 *
              /SV3 * /SV2 * SV1 * IMBRDY * BGT
            + /RSTCPU * /IDISB * MKEN * MCACHE * /CLEN1 * CLEN0 *
              /SV3 * SV2 * /SV1 * IMBRDY * BGT
            + /RSTCPU * IDISB * ( SV3 + SV2 + SV1 )

IDISB.CLKF    = ACLK

;-------------------------------------------------------------------
;This state machine MBLAST for the memory bus. MBLAST is active on
;the last transaction on the memory bus.This state machine also
;generates the burst address lines MA2 and MA3.
;
;It also generates BRDY disable signal IBRDY used when caching is
;done with MRO attribute.In this case BRDY is generated only for
;the no. of transfers required by the CPU.
;
;The state machine also generates KWEND and SWEND to 82495. MKEN is
;generated by address decoding on the debug board. It is valid one
;clock after MADS. KWEND is generated at that time.
;
;-------------------------------------------------------------------
```

241166-34

2

8.0 APPENDIX

8.1 Configuration during RESET

The following table lists the configuration inputs used during reset.

Table 8-1. Configurations Options Set During Reset

82495DX Input	Input Source (state machine, etc.)	Input Value During Reset	Explanation
CFG3	Tied low	0	82495DX initialization
KWEND# [CFG2]	XWND	0	82495DX initialization
SWEND# [CFG1]	XWND	0	82495DX initialization
CNA# [CFG0]	Tied high	1	82495DX initialization
CPLOCK# [CPLOCKEN]	XKEN PLD	Depends on MBC signal CPLOCKEN (J13)	Enables/disables PLOCK functionality
BGT# [CLDRV]	XMADS	1	Select low capacitance drivers for 82495DX
CRDY# [SLFTST]	XCRDY	Depends on MBC signal SLFTST# (J2) and MAHOLD from host system.	If MAHOLD is asserted, CPU-Cache self-test is executed following reset.
MALE[WWOR#]	XMBReq	0	Select weak write ordering
MBALE[HIGHZ#]	Tied high	1	Required for selftest
MFRZ# [MDLDRV]	XMEOC PLD	1	Normal capacitance data bus output drivers
SYNC# [MALDRV]	Tied high	1	Normal address bus output drivers
MSEL# [MTR4/MTR8#]	XMADS	1	Four transfers per Mbus transaction
MZBT# [MX4/MX8#]	Tied high	1	Four I/O pins per 82490DX.
SNPCLK[SNPMD]	Jumper 14	high or low	Select either synchronous or strobed snoops

8.2 Worse Case Timing Analysis

8.2.1 CLOCK SKEWS

- The clock skew from PCLK to CLK[1:4] is calculated from the GA1210 data sheet. It works out to be SkewA = ±1 ns. Similarly for PCLK to CLK[5:6]. Therefore, the worse case skew from the Module's 50 MHz clocks to the MBC's 50 MHz clocks is ±2 ns.
- DMCLK is nominally delayed 2 ns from MCLK. The worse case clock skew can be calculated from the GA1110 data sheet. It works out to be SkewB = 1.5 ns min/2.5 ns max.

8.2.2 MODULE INPUT SIGNALS

- BRDY#, CRDY#: Required Setup = 8 ns, Hold = 3 ns.

  To satisfy the CPU-Cache module's input hold time, the BRDY# and CRDY# signals are wrapped around inside the PAL16R4 in order to increase the minimum valid delay time.

  Setup = Clock Period − SkewA − PAL16R4__5 t_{CO} max − PAL16R4__5 t_{PD} max

  = 20 − 2 − 4 − 5 = 9 ns

  Hold = PAL16R4__5 t_{CO} min + PAL16R4__5 t_{PD} min − SkewA

  = 3 + 3 − 2 = 4 ns

- KWEND#, SWEND#, BGT#: Required Setup = 6 ns, Hold = 3 ns.

  In this case the hold time is more easily met by continuing to assert these signals past the CLK cycle in which they are sampled by the CPU-Cache module.

  Setup = Clock Period − SkewA − 85C060 t_{CO} max (asynchronous mode)

  = 20 − 2 − 12 = 6 ns

- MRO#, MKEN#

  From the schematics and PLD codes, MKEN# generation experiences three propagation delays from the cache address—twice through a 85C224 and once through a 20L8__5. The maximum delay allowed was calculated in Section 5.3.4.1 to be 63 ns. After accounting for clock skew, this reduces to 61 ns.

  Tdecode = 2 * 85C224 t_{PD} max + PAL20I8__5 t_{PD} max

  = 2 * 10 ns + 5 ns = 25 ns

- MEOC#: Required Setup = 4 ns, Hold = 3 ns.

  Setup = Clock Period − SkewB max − PAL20L8__5 t_{PD} max

  = 20 − 2.5 − 5 = 12.5 ns

  Hold = PAL20L8__5 t_{PD} min + SkewB min

  = 3 + 1.5 = 4.5 ns

- MBRDY490#: Required Setup = 5 ns, Hold = 2 ns

  Setup = Clock Period − SkewB max − PAL20L8__5 t_{PD} max

  = 20 − 2.5 − 5 = 12.5 ns

  Hold = PAL20L8__5 t_{PD} min + SkewB min

  = 3 + 1.5 = 4.5 ns

8.2.3 MODULE OUTPUT SIGNALS

The following signals are used as inputs by the 85C060's in asynchronous clocked mode in which the setup and hold times ae 2 ns and 2 ns respectively.

- CADS#, SNPADS#, CDTS#:

  Setup = Clock Period − SkewA − Module Valid Delay max (t_{32})

  = 20 − 2 − 12 = 6 ns

  Hold = Module Valid Delay min (t_{32}) − SkewA

  = 4 − 2 = 2 ns

- CLEN1, CLEN0, MCACHE#, RDYSRC, PALLC:

  Easily satisfies setup since these signals are sampled after the CLK period in which they are valid. Hold requirement is met since these signals remain valid throughout the bus cycle.

- HLDA#:

  Setup = Clock Period − SkewA − Module Valid Delay max (t_{17})

  = 20 − 2 − 14 = 4 ns

  Hold time is not important since the active edge is not used in the MBC.

9.0 SUGGESTED OTHER READING

Documents related to this Applications Note are listed as follows:

Clock Design in 50 MHz Intel486™ Systems, AP-453 (Order Number 241082-001)

Cache Tutorial 1991 (Order Number 296543-002)

An Application Note on 50 MHz Intel486™ DX CPU-Cache Module Thermal Performance/Limits/Requirements, AP-455 (Order Number 241089-001)

Designing a Memory Bus Controller for a 50 MHz Intel486™ DX CPU-Cache Write-Through EISA System, AP-460 (Order Number 241196-001)

50 MHz Intel486™ DX CPU-Cache Subsystem Architectural Overview 82495DX/82490DX (Order Number 241085-001)

50 MHz Intel486™ DX CPU-Cache Module Hardware Reference Manual (Order Number 241091-001)

intel

AP-460

APPLICATION NOTE

Designing a Memory Bus Controller for a 50 MHz Intel486™ DX CPU-Cache Write-Through EISA System

TAUFIK T. MA
TECHNICAL MARKETING ENGINEER

September 1991

For the complete data sheet on this device, contact Intel's Literature Distribution Department (800) 548-4725.

PRELIMINARY

Order Number: 241196-001

APPLICATION NOTE

Cache and Memory Design Considerations for the Intel486™ DX2 Microprocessor

2

TAUFIK T. MA
INTEL TECHNICAL MARKETING

May 1992

PRELIMINARY
Order Number: 241261-001

Cache and Memory Design Considerations for the Intel486™ DX2 Microprocessor

CONTENTS

PAGE

1.0 INTRODUCTION 2-1009

2.0 THE HIGH-PERFORMANCE Intel486™ DX2 MICROPROCESSOR 2-1009

2.1 Internal Cache Hit Rates 2-1010

2.2 Bus Cycle Mix 2-1011

2.3 Bus Utilization 2-1012

2.4 Profiles of Some Applications ... 2-1013

2.5 Wait States Explained 2-1014

2.5.1 The Ideal Zero Wait State Memory System 2-1014

2.5.2 Adding Wait States 2-1015

2.5.3 Wait States and CPU Stalls 2-1020

2.5.3.1 Delay Till First Ready of a Read 2-1020

2.5.3.2 Wait states on Bursts 2-1020

2.5.3.3 Write Wait States 2-1021

3.0 MEMORY DESIGN OPTIMIZATION 2-1022

3.1 Page Mode DRAM 2-1023

3.2 Interleaving 2-1023

CONTENTS

PAGE

3.3 Memory Read Performance Considerations 2-1024

3.4 Memory Write Performance Considerations 2-1025

3.5 Viability of Intel486 DX2 System without an External Cache 2-1032

4.0 CACHE DESIGN OPTIMIZATION 2-1032

4.1 Overall Effect of an External Cache on CPU Performance 2-1033

4.2 Effect of Cache Size and Associativity 2-1034

4.3 Improving the Performance of a Write-Through Cache 2-1036

4.3.1 Memory Write Pipelining ... 2-1036

4.3.2 External Write Buffers 2-1036

4.3.3 Performance with an External Write-Through Cache 2-1038

4.4 Write-Back Caches 2-1039

4.4.1 Main Memory Controller Considerations 2-1039

4.4.2 Write-Back Cycle 2-1039

5.0 CONCLUSION 2-1041

CONTENTS

PAGE

FIGURES

Figure 2.1 The 66MHz Intel486 DX2 CPU Internal Architecture 2-1009

Figure 2.2 The Intel486 DX2 Microprocessor is More Sensitive to Memory Latency than Its Predecessor 2-1010

Figure 2.3 There will be a Range of L1 Hit Rates for Different Applications/Operating Systems (%) 2-1011

Figure 2.4 The Effect of L1 Cache Hits on the External Bus Cycle Mix for an Integer SPEC Benchmark 2-1011

Figure 2.5 Definition of Bus Utilization 2-1012

Figure 2.6 Memory Performance Notation 2-1014

Figure 2.7 Intel486 DX2 CPU Performance Degradation as Wait States are Added - for the SPEC1 Application Trace (UNIX) 2-1016

Figure 2.8 Intel486 DX CPU Performance Degradation as Wait States are Added - for the SPEC1 Application Trace (UNIX) 2-1016

Figure 2.9 Intel486 DX2 CPU Performance Degradation as Wait States are Added - for the GCC Application Trace (UNIX) 2-1017

Figure 2.10 Intel486 DX CPU Performance Degradation as Wait States are Added - for the GCC Application Trace (UNIX) 2-1017

Figure 2.11 Intel486 DX2 CPU Performance Degradation as Wait States are Added - for the Pagemaker Application Trace (Windows) 2-1018

Figure 2.12 Intel486 DX CPU Performance Degradation as Wait States are Added - for the Pagemaker Application Trace (Windows) 2-1018

Figure 2.13 Intel486 DX2 CPU Performance Degradation as Wait States are Added - for the Turbo C Application Trace (DOS) 2-1019

CONTENTS

PAGE

Figure 2.14 Intel486 DX CPU Performance Degradation as Wait States are Added - for the Turbo C Application Trace (DOS) 2-1019

Figure 2.15 A Zero-Wait State Write for the Intel486 DX2 CPU 2-1020

Figure 2.16 The Intel486 DX2 CPU's Write Buffers are More Heavily Used than the Intel486 DX CPU's 2-1021

Figure 2.17 Reads Stalled as a Result of a Write Already In Progress 2-1022

Figure 3.1 The Two Cacheless Intel486 DX2 CPU-Based Systems Considered 2-1022

Figure 3.2 A Page Mode Burst Read (Page Hit on Lead-Off Cycle) 2-1023

Figure 3.3 Burst Read Cycle from a Paged-Interleaved Memory System 2-1024

Figure 3.4 Total Intel486 DX2 CPU Execution Time versus Memory Read Performance - for SPEC1 (UNIX) 2-1024

Figure 3.5 Total Intel486 DX2 CPU Execution Time versus Memory Read Performance - for Pagemaker (Windows) 2-1025

Figure 3.6 Total Intel486 DX2 CPU Execution Time versus Memory Read Performance - for Turbo C (DOS) 2-1025

Figure 3.7 Page Mode DRAM Allow for Fast Back-to-back Writes 2-1026

Figure 3.8 Adding One Write Buffer to the Memory System 2-1026

Figure 3.9 Pipelining the Writes to Memory 2-1027

Figure 3.10 A Data Latch is Required for Write Buffering or Pipelining 2-1027

Figure 3.11 Total Execution Time versus Write Performance- for SPEC1 (UNIX) 2-1028

Figure 3.12 Total Execution Time versus Write Performance - for Pagemaker (Windows) 2-1029

CONTENTS PAGE

Figure 3.13 Total Execution Time versus Write Performance - for Turbo C (DOS) 2-1029

Figure 3.14 Adding Write Buffers Does Not Improve Stalls Because of Reads on Busy Writes .. 2-1031

Figure 4.1 Different Cache Architectures Discussed 2-1032

Figure 4.2 Adding an External Cache Decreases Execution Time - for SPEC1 (UNIX) 2-1033

Figure 4.3 Adding an External Cache Decreases Execution Time - for Pagemaker (Windows) 2-1033

Figure 4.4 Adding an External Cache Decreases Execution Time - for Turbo C (DOS) 2-1034

Figure 4.5 Total Execution Time for the Intel486 DX2 CPU as a Function of Cache Size and Associativity - for SPEC1 .. 2-1035

Figure 4.6 L2 Read Hit Rate as a Function of Cache Size and Associativity (UNIX) 2-1035

Figure 4.7 Adding External Write Buffers to an External Cache Reduces Execution Time 2-1037

Figure 4.8 A Hierarchy of Caches and Write Buffers 2-1038

Figure 4.9 Improving the Write Performance Benefits a Write Through Cache - for SPEC1 2-1038

Figure 4.10 Different Architectures will Effect CPU Performance with a Write-Back Cache 2-1040

Figure 4.11 Different Implementations of the Write-Back Cycle 2-1040

Figure 4.12 Intel486 DX2 CPU Total Execution Time with Different Cache Size, Associativity, Memory Speed and Write-Back Method - for SPEC1 2-1041

CONTENTS PAGE

TABLES

Table 2.1 Bus Profiles of UNIX Applications with a Zero Wait-State Memory System 2-1013

Table 2.2 Bus Profiles of Windows Applications with a Zero Wait-State Memory System 2-1013

Table 2.3 Bus Profiles of DOS Applications with a Zero Wait-State Memory System 2-1013

Table 2.4 Number of Internal Clocks (millions) Needed to Complete Application Trace 2-1015

Table 2.5 Total Execution Time in Seconds 2-1015

Table 2.6 CPU Performance versus Total Execution Time 2-1015

Table 2.7 Percentage of Total Execution Time Stalled under the Three Different Write Stall Conditions 2-1022

Table 3.1 Page Hit Ratios 2-1023

Table 3.2 Memory Systems used for the No-Cache System Test 2-1024

Table 3.3 Memory Systems with Different Write Performances 2-1028

Table 3.4 Stall Statistics for the Write Buffers for the SPEC1 (UNIX) Trace 2-1030

Table 4.1 Memory Systems for the Write-Through Cache Test 2-1034

Table 4.2 Memory Systems Used for Write-Through Cache Test .. 2-1038

Table 4.3 Page Hit Ratios for a Write-Back Cache - for SPEC1 2-1039

Table 4.4 Memory Systems used for Write-Back Cache Test 2-1041

1.0 INTRODUCTION

This section discusses CPU performance optimization techniques for the Intel486™ DX2 microprocessor. The reader should be familiar with the Intel486™ DX microprocessor as well as knowledgeable about memory systems and cache architectures. For further reference, the reader is directed to the following documents and application notes (corresponding Intel order number is shown in parentheses):

- Intel486™ DX2 Microprocessor Data Book (241245-001)
- Intel486™ DX Microprocessor Data Book (240440-004)
- Intel486™ DX Microprocessor Hardware Reference Manual (240552-001)
- Cache Tutorial 1991 (296543-002)
- AP447: A Memory Subsystem for the Intel486™ Family of Microprocessors including Second Level Cache (240799-002)
- 485TurboCache Module Intel486™ DX Microprocessor Cache Upgrade (240722-002)

2.0 The High-Performance Intel486™ DX2 Microprocessor

The Intel486 DX2 Microprocessor is functionally equivalent to the now-familiar Intel486™ DX Microprocessor. However, the Intel486 DX2 CPU's internal core runs at twice the frequency of its external bus. This architecture enables a very high level of performance while, at the same time, maintaining straightforward system design.

The Intel486 DX2 CPU is partitioned such that the cache and write buffers operate at the full core speed as illustrated in Figure 2.1. As such, the processor is only slowed to the external bus speed on cache misses and when the write-buffers are full. The Intel486 DX2 CPU's external bus interface is identical to its predecessor, i.e. all system cycles emanating from the CPU look exactly as if they would from the Intel486 DX CPU. The Intel486 DX2 microprocessor includes additional features such as JTAG boundary scan and power-down capability that are not covered in this section.

2

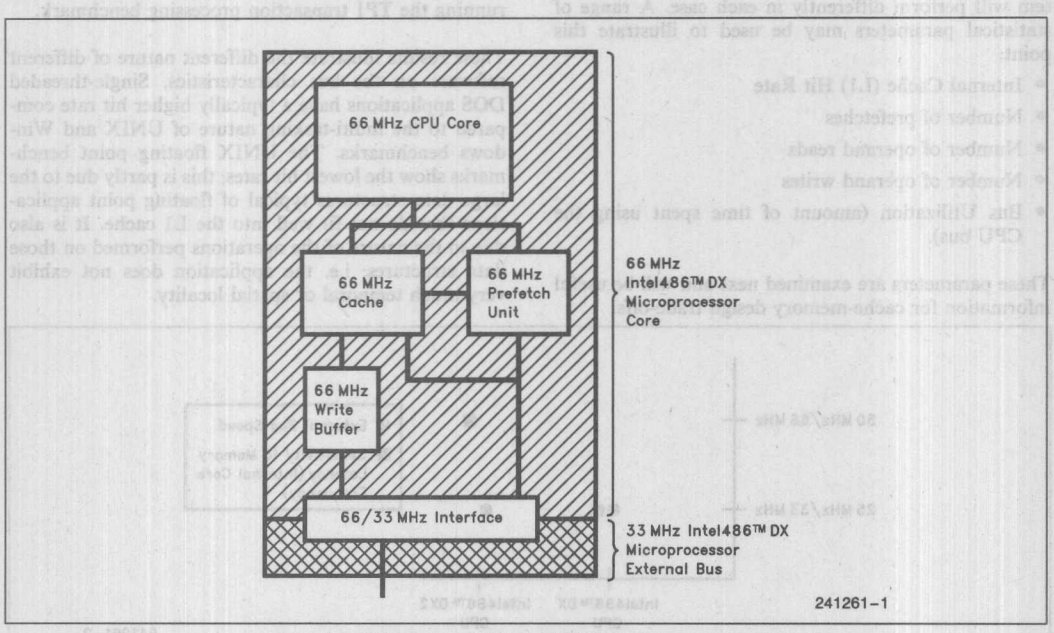


Figure 2.1. The 66MHz Intel486™ DX2 CPU Internal Architecture

MS-DOS, Windows, Word, Excel and Flight Simulator are trademarks of Microsoft Corporation.
UNIX is a trademark of UNIX System Laboratories.
Lotus, 123 are trademarks of Lotus Development Corporation.
Turbo C is a trademark of Borland International.
AutoCAD is a trademark of AutoDesk, Inc.

Performance optimization for the Intel486 DX2 CPU is subtly different than for the Intel486 DX CPU due to the difference in the internal architecture. This is evident if you consider that external memory latencies now affect the full speed core by twice as many CPU clocks (refer to Figure 2.2). In other words, the memory system should really be designed to satisfy the data throughput demands of a 50/66MHz CPU. The next few sections examine the situation in detail so that educated trade-offs can be made during system design. The discussions will focus on CPU-cache memory performance; I/O performance and other architectural issues are not addressed in this applications note.

In an ideal system, all CPU cycles operate at zero-wait states and the theoretical maximum performance of the CPU is achieved. However, short of spending a lot of money on SRAMs, a real system always falls short of the ideal zero wait-states. There are many cache-memory designs that differ in both architecture and implementation. However, it may be impossible to design a system that performs better than all others across all applications; different applications generate different types of CPU bus activity and the cache-memory system will perform differently in each case. A range of statistical parameters may be used to illustrate this point:

- Internal Cache (L1) Hit Rate
- Number of prefetches
- Number of operand reads
- Number of operand writes
- Bus Utilization (amount of time spent using the CPU bus).

These parameters are examined next and will be useful information for cache-memory design trade-offs.

2.1 Internal Cache Hit Rates

The Intel486 DX2 CPU maintains the same unified code/data, four-way interleaved, 8K-byte internal cache as the Intel486 DX CPU. The internal cache (L1) hit rate is shown in Figure 2.3 for some different operating systems and applications. These hit rates were obtained from instruction traces captured from specific applications. Note that the L1 hit rate for the Intel486 DX2 CPU will be almost identical to that for the Intel486 DX CPU; the 2X-internal frequency does not significantly alter the cache miss statistics.

The DOS applications included Auto Cad, Lotus123, Excel, Turbo C, and Flight Simulator. Lotus123 had the highest hit rate at 99% while Auto Cad was the lowest at 89.6%. The Windows 3.0 results included instructions executed while clipping an image, drawing a dialog box, executing Excel and while executing Pagemaker. The category UNIX-iSPEC refers to applications within the UNIX SPEC benchmark suite that are mostly integer intensive, whereas UNIX-fSPEC refers to those which are floating point intensive. The last category UNIX-TP1 refers to the hit rates typical while running the TP1 transaction processing benchmark.

These results illustrate the different nature of different software on the bus characteristics. Single-threaded DOS applications have a typically higher hit rate compared to the multi-tasking nature of UNIX and Windows benchmarks. The UNIX floating point benchmarks show the lowest hit rates; this is partly due to the large data structures typical of floating point applications that do not fit well into the L1 cache. It is also due to the nature of the operations performed on those data structures; i.e. the application does not exhibit very much temporal or spatial locality.

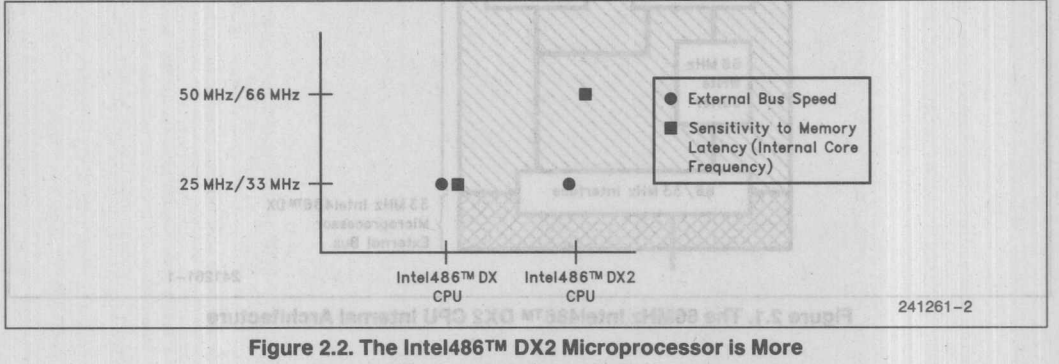


Figure 2.2. The Intel486™ DX2 Microprocessor is More Sensitive to Memory Latency than Its Predecessor

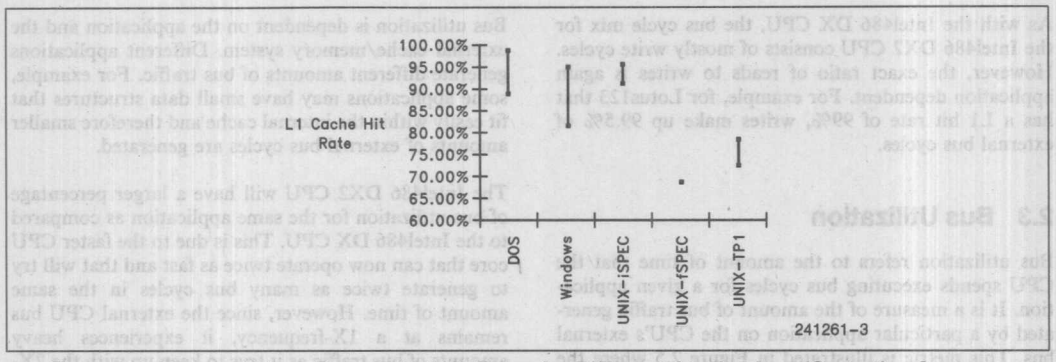


Figure 2.3. There will be a Range of L1 Hit Rates for Different Applications/Operating Systems

TP1 is a UNIX multiuser multithreaded application with heavy amounts of disk and I/O as well as computation. The L1 cache hit rate is low since it has many active contexts due to multiple requests per user.

High L1 hit rates are typical of many prevalent and commonly used DOS benchmarks - some have hit rates approaching 100%. This is unfortunate since they fail to properly account for the more realistic external cache and memory demands of most DOS and Windows applications. These demands will continue to be true with the advent of newer graphical-user-interface-based multi-tasking operating systems and applications. With these benchmarks, there is a danger of misrepresenting system performance for the Intel486 DX microprocessor. For the Intel486 DX2 microprocessor, this misrepresentation can be even more damaging since L1 cache misses incur a penalty that is twice the number of external clock cycles - since the CPU core now runs twice as fast internally. In other words, the Intel486 DX2 CPU is twice as sensitive to wait states compared to the Intel486 DX CPU. To properly gauge the external cache and memory performance, more demanding benchmarks (e.g. UNIX SPECmarks) or real application benchmarks should be used.

2.2 Bus Cycle Mix

Different applications cause the CPU to generate a different number of reads, writes and instruction prefetches. The reads and prefetches are filtered by the internal L1 cache before reaching the external CPU bus. All writes propagate through to the external bus since the internal cache follows a write-through protocol. This is shown in Figure 2.4 for an instruction trace captured from an integer SPEC benchmark.

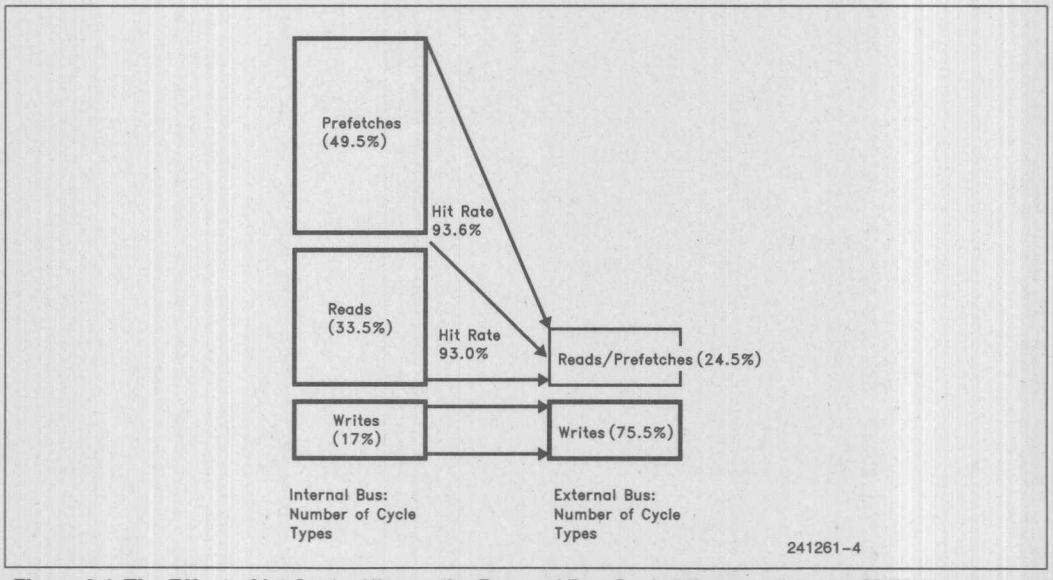


Figure 2.4. The Effect of L1 Cache Hits on the External Bus Cycle Mix for an Integer SPEC Benchmark

As with the Intel486 DX CPU, the bus cycle mix for the Intel486 DX2 CPU consists of mostly write cycles. However, the exact ratio of reads to writes is again application dependent. For example, for Lotus123 that has a L1 hit rate of 99%, writes make up 99.5% of external bus cycles.

2.3 Bus Utilization

Bus utilization refers to the amount of time that the CPU spends executing bus cycles for a given application. It is a measure of the amount of bus traffic generated by a particular application on the CPU's external bus. This metric is illustrated in Figure 2.5 where the CPU bus is busy for 75% of the twelve external clock cycles shown.

Bus utilization is dependent on the application and the external cache/memory system. Different applications generate different amounts of bus traffic. For example, some applications may have small data structures that fit easily within the internal cache and therefore smaller amounts of external bus cycles are generated.

The Intel486 DX2 CPU will have a larger percentage of bus utilization for the same application as compared to the Intel486 DX CPU. This is due to the faster CPU core that can now operate twice as fast and that will try to generate twice as many bus cycles in the same amount of time. However, since the external CPU bus remains at a 1X-frequency, it experiences heavy amounts of bus traffic as it trys to keep up with the 2X-internal core. In other words, with faster internal core execution, less time is spent idling on the external bus.

Different external cache/memory systems also affect the bus utilization; faster cache/memory systems allow CPU cycles to complete faster and therefore free up the bus more.

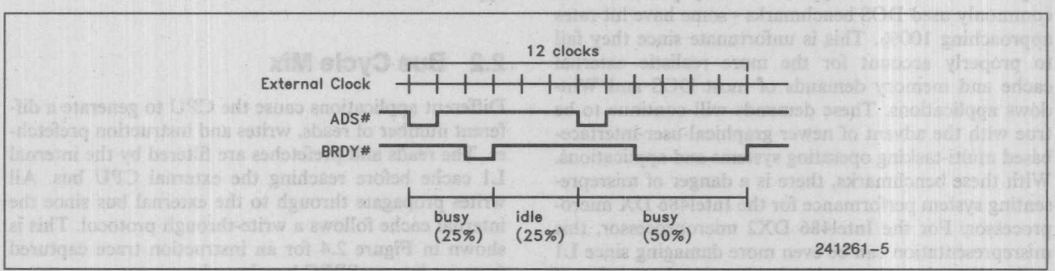


Figure 2.5. Definition of Bus Utilization

2.4 Profiles of Some Applications

The Intel486 DX2 CPU bus characteristics of some different applications and operating systems are shown in Tables 2.1 through 2.3. These results are derived from traces captured from the actual applications. These traces were subsequently used in a CPU-cache-memory simulator to extract the desired information. The results shown here assume an ideal zero wait-state memory system; i.e. all bus cycles complete in zero wait-states.

Table 2.1. Bus Profiles of UNIX Applications with a Zero Wait-State Memory System

UNIX Applications	SPEC1	GCC	UNIXMIX1
Total Number of CPU clocks simulated	12.14M	12.54M	12.44M
Overall L1 hit rate	91.4%	90.5%	94.3%
Prefetch hit rate	93.6%	91.7%	94.8
Read hit rate	93.0%	90.3%	94.2%
Write hit rate	82.0%	85.7%	93.5%
Number of external bus cycles	1.12M	0.882M	1.18M
% bus code prefetches	14.5%	22.6%	10.2%
% bus data reads	10.0%	20.0%	8%
% bus data writes	75.5%	57.4%	81.8%
Bus Utilization	51.6%	46.4%	51.4%

Table 2.2. Bus Profiles of Windows Applications with a Zero Wait-State Memory System

Windows Applications	Word	Excel—Calc	Pagemaker
Total Number of CPU clocks simulated	27.02M	7.39M	30.06M
Overall L1 hit rate	95.2%	78.4%	88.1%
Prefetch hit rate	96.9%	76.0%	81.8%
Read hit rate	98.0%	85.7%	95.2%
Write hit rate	87.7%	69.7%	83.5%
Number of external bus cycles	2.43M	0.654M	3.04M
% bus code prefetches	4.2%	27.9%	19.3%
% bus data reads	3.4%	15.1%	6.7%
% bus data writes	92.4%	57.0%	74%
Bus Utilization	40.9%	60.1%	56.0%

Table 2.3. Bus Profiles of DOS Applications with a Zero Wait-State Memory System

DOS Applications	Excel	Turbo C	Auto Cad
Total Number of CPU clocks simulated	11.1M	13.9M	16.1M
Overall L1 hit rate	98.2%	95.6%	89.3%
Prefetch hit rate	98.8%	94.2%	87.7%
Read hit rate	97.9%	98.1%	96.5%
Write hit rate	97.4%	93.8%	81.3%
Number of external bus cycles	1.06M	1.18M	1.77M
% bus code prefetches	2.1%	11.2%	11.8%
% bus data reads	3.1%	3.0%	4.0%
% bus data writes	94.8%	85.8%	84.2%
Bus Utilization	41.0%	40.6%	55.5%

The UNIX applications are described as:

- SPEC1: A mixture of integer SPEC benchmark suite programs running concurrently.
- GCC: SPEC benchmark suite GNU C compiler, compiling itself.
- UNIXMIX1: A mixture of UNIX utility programs like awk and grep, running concurrently.

The Microsoft Windows 3.0 applications are described as:

- Word: Microsoft Word for Windows converting a document for import (no VGA activity, includes kernel calls).
- Excel—Calc: Microsoft Excel for Windows running a calculation.
- Pagemaker: Pagemaker for Windows formatting a document (no VGA activity, includes kernel calls)

The three DOS applications shown are described as:

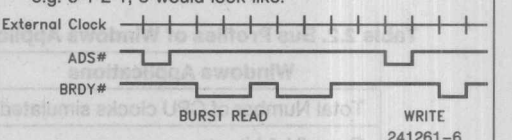
- Excel: Microsoft Excel (DOS version) recalculating a spreadsheet.
- Turbo C: Borland's Turbo C compiler compiling a large C program.
- Auto Cad: Auto Desk's Auto Cad program computing and displaying a drawing (reason for low hit rate)

Note that most DOS applications will typically use the external bus much less than UNIX or Windows applications. On the other hand, heavy duty DOS applications such as Auto Cad will actually exhibit a larger demand for the external bus similar to the demands of UNIX and Windows. Also, recall that the bus utilization numbers shown assume a zero wait-state memory system; more realistic external cache/memory systems with a finite number of wait-states will actually experience a larger percentage of bus utilization. Finally, note that since the L1 hit rate for the DOS applications is high, the external bus mix consists of mostly writes.

2.5 Wait States Explained

Now that we have considered the characteristics of L1 hit rates, bus cycle mix and utilization, we can examine the impact that the cache-memory system has on overall Intel486 DX2 CPU performance. To examine these effects, traces were captured from three applications (one from each operating environment) and were used in a CPU-cache-memory simulator to measure the CPU performance under different conditions. The traces used were SPEC1 (UNIX), Pagemaker (Windows) and Turbo C (DOS). The simulator used is an accurate and convenient method of comparing performance by varying different parameters independently. This allows us to develop some heuristic rules to guide system design. Before continuing, the following convention is defined to denote memory performance:

Notation Convention:
A memory system's performance is abbreviated as:
Lead-off clocks - Burst 2 - Burst 3 - Burst 4, Write clocks
e.g. 3-1-2-1, 3 would look like:



A zero-wait-state case corresponds to a 2-1-1-1, 2 memory system.

Figure 2.6. Memory Performance Notation

2.5.1 THE IDEAL ZERO WAIT STATE MEMORY SYSTEM

As a starting point, the ideal zero-wait-state memory system was characterized for three applications. For I/O cycles, it was assumed that a constant 8 wait-states were required. Since the I/O instructions were a small portion of the instruction traces used, any inaccuracy due to this assumption will be insignificant.

The total execution times reported are as follows:

Table 2.4. Number of Internal Clocks (millions) Needed to Complete Application Trace

	SPEC1	Pagemaker	Turbo C
Intel486 DX CPU	10.76	26.68	13.34
Intel486 DX2 CPU	12.14	30.06	13.90

As an example, we can now compare the actual time required to complete the applications between a 66MHz Intel486 DX2 CPU and a 33MHz Intel486 DX CPU. The number of clock cycles is multiplied by 15ns for the Intel486 DX2 CPU and by 30ns for the Intel486 DX CPU.

Table 2.5 Total Execution Time in Seconds

	SPEC1	Pagemaker	Turbo C
Intel486 DX CPU	323 ms	800 ms	400 ms
Intel486 DX2 CPU	182 ms	451 ms	209 ms
Performance Increase	+77%	+77%	+91.4%

Note that although the Intel486 DX2 CPU's internal clock rate is twice that of the corresponding Intel486 DX CPU, the relative improvement is less than 100%. This is due to cache miss reads and write cycles that run at the external bus speed. Note that the improvement is much greater for Turbo C which has a lower cache miss rate and low bus utilization.

2.5.2 ADDING WAIT STATES

As wait states are added to the ideal zero wait state memory system, performance degrades. Three memory parameters are of interest in characterizing the memory performance. They are:

- Number of wait states added to the first ready of a read (a.k.a. the lead-off cycle). e.g. One wait-state with zero wait-state burst = 3-1-1-1
- Number of wait states during the remainder of the burst cycle. e.g. a zero wait-state lead-off with a one wait-state burst = 2-2-2-2
- Number of wait states on a write cycle. e.g. a one wait-state write takes three clocks.

To examine the impact of adding wait-states to each of the parameters above, three series of simulations are done where wait states are added to each memory parameter separately while the other memory parameters are held constant:

	Zero wait-states	1 wait-state	2 wait-states	3 wait-states
• Lead-off Series:	2-1-1-1, 2	3-1-1-1, 2	4-1-1-1, 2	5-1-1-1, 2 . . .
• Burst Series:	2-1-1-1, 2	2-2-2-2, 2	2-3-3-3, 2	2-4-4-4, 2
• Write Series:	2-1-1-1, 2	2-1-1-1, 3	2-1-1-1, 4	2-1-1-1, 5 . . .

As the number of wait states increases, the number of clocks needed to complete the application increases (and the CPU performance decreases). This series of measurements is used to separate out the dependency of the CPU performance on the different memory parameters. This information is useful for subsequent external cache/memory design trade-offs.

The number of clocks needed to complete the application relative to the zero wait-state case is referred to as the relative total execution time. This metric will be used in the following graphs instead of the reciprocal of total execution time which would be CPU performance; this is so that any inherent linear relationships between wait-states and execution time can be more easily recognized. To translate the total execution time back to CPU performance, the following table is provided for convenience (100% refers to the zero wait-state case):

Table 2.6. CPU Performance versus Total Execution Time

Total Execution Time	100.00%	110.00%	120.00%	130.00%	140.00%	150.00%	160.00%
CPU Performance	100.00%	90.91%	83.33%	76.92%	71.43%	66.67%	62.50%

Figures 2.7 and 2.8 show the total execution time as wait states are added for the SPEC1 trace described earlier.

As would be expected, as wait states are added, the relative total execution time for the Intel486 DX2 CPU increases faster than the Intel486 DX CPU. (however, note that the absolute total execution time for a Intel486 DX2 CPU will never be greater than a Intel486 DX CPU of the same external bus speed). Also note that the order of importance of the memory parameters changes between the Intel486 DX and the Intel486 DX2 CPU. For the Intel486 DX CPU, burst performance is the most important, with read-lead-off performance being slightly more important than writes. This conclusion was presented in Chapter 4 of the original Intel486 DX Microprocessor Hardware Reference Manual (Order Number 240552-001). However for the Intel486 DX2 CPU, while burst performance is still the most important, write performance becomes more important than read-lead-off performance.

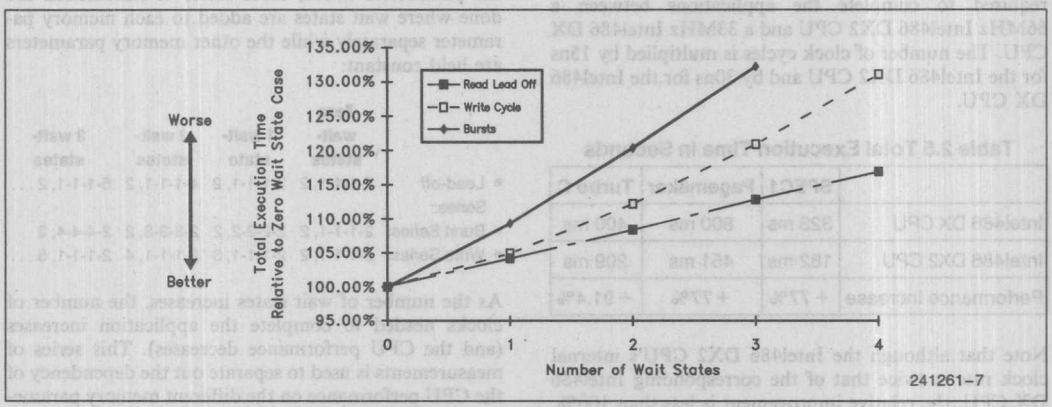


Figure 2.7. Intel486 DX2 CPU Performance Degradation as Wait States are Added - for the SPEC1 Application Trace (UNIX)

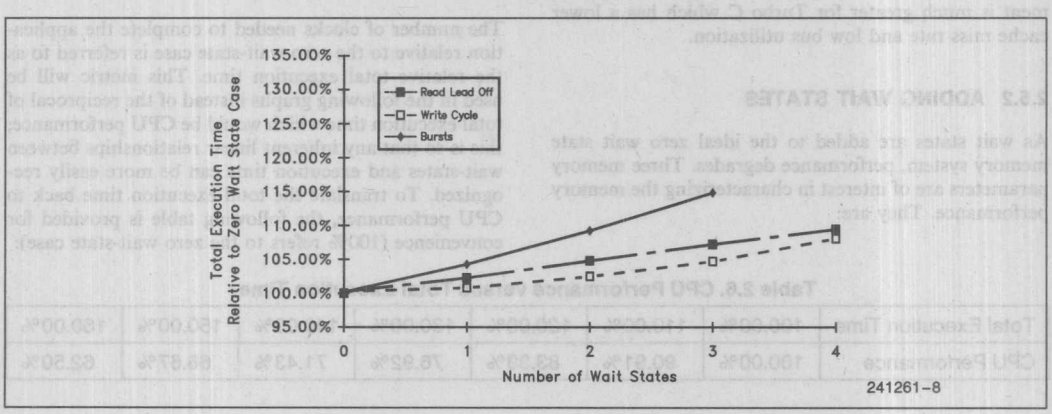


Figure 2.8. Intel486 DX CPU Performance Degradation as Wait States are Added - for the SPEC1 Application Trace (UNIX)

Figures 2.9 and 2.10 show the total execution time as wait states are added for the GCC trace described earlier.

Compared to the results for the SPEC1 application trace, the GCC results with the Intel486 DX2 CPU still show the burst cycles as being the most sensitive to wait states. However, note that the lead-off cycle is now more important than write cycles. This is due to the small percentage of bus writes (57.4%) while running the GCC application as compared with the SPEC1 trace.

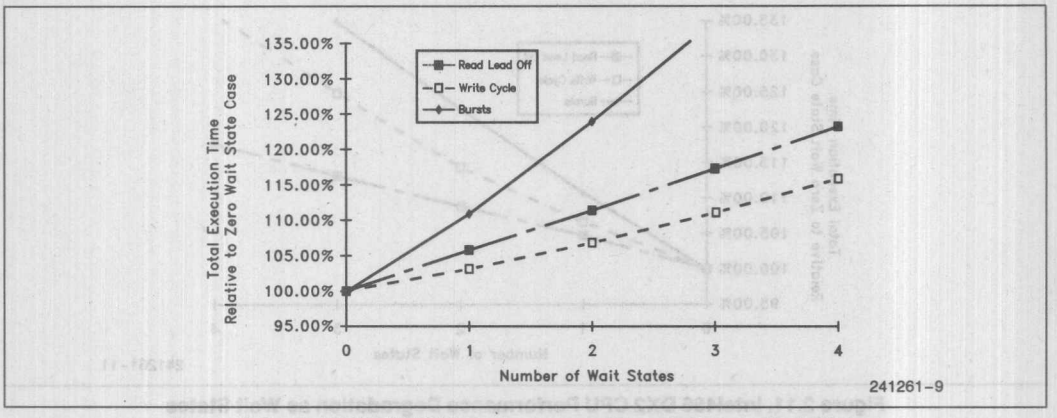


Figure 2.9. Intel486 DX2 CPU Performance Degradation as Wait States are Added - for the GCC Application Trace (UNIX)

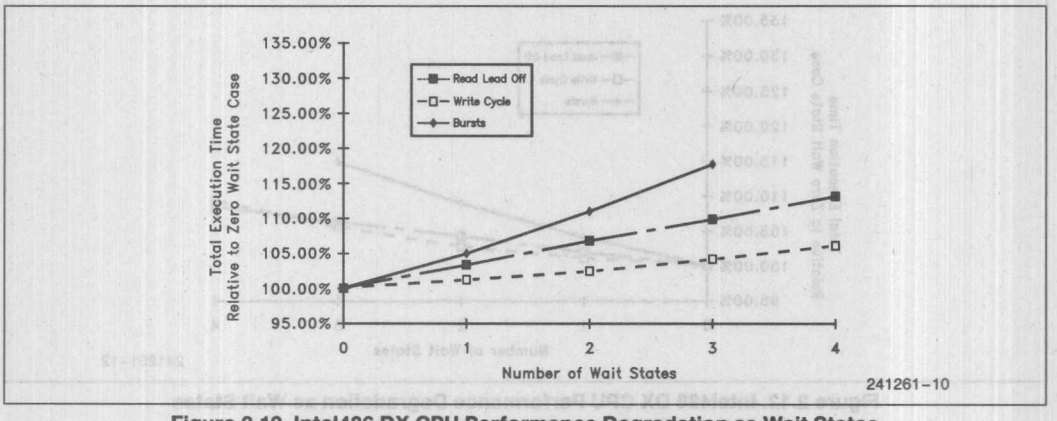


Figure 2.10. Intel486 DX CPU Performance Degradation as Wait States are Added - for the GCC Application Trace (UNIX)

The results for the Pagemaker Application under Windows are shown in Figures 2.11 and 2.12.

The same conclusions can be made for the Pagemaker application under Windows as for the SPEC1 results earlier. Basically, the degradation of the Intel486 DX2 CPU's relative execution time increases faster as wait-states are added as compared to the Intel486 DX CPU.

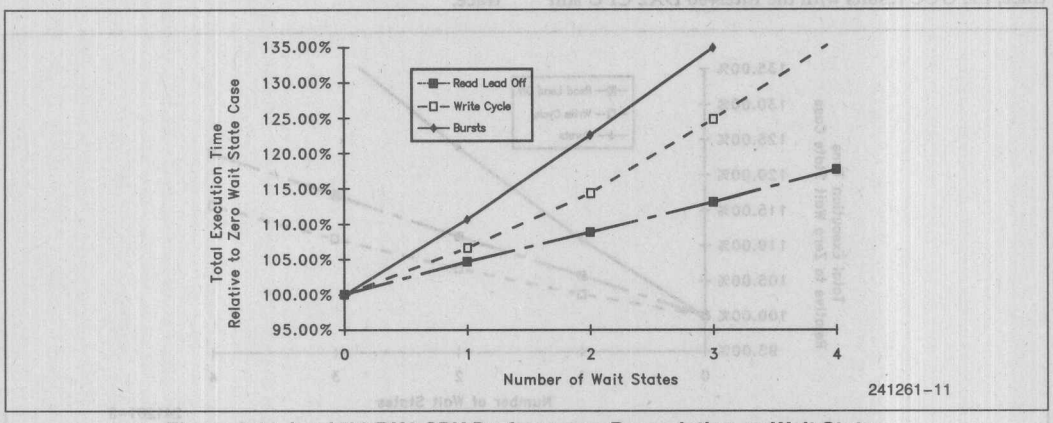


Figure 2.11. Intel486 DX2 CPU Performance Degradation as Wait States are Added - for the Pagemaker Application Trace (Windows)

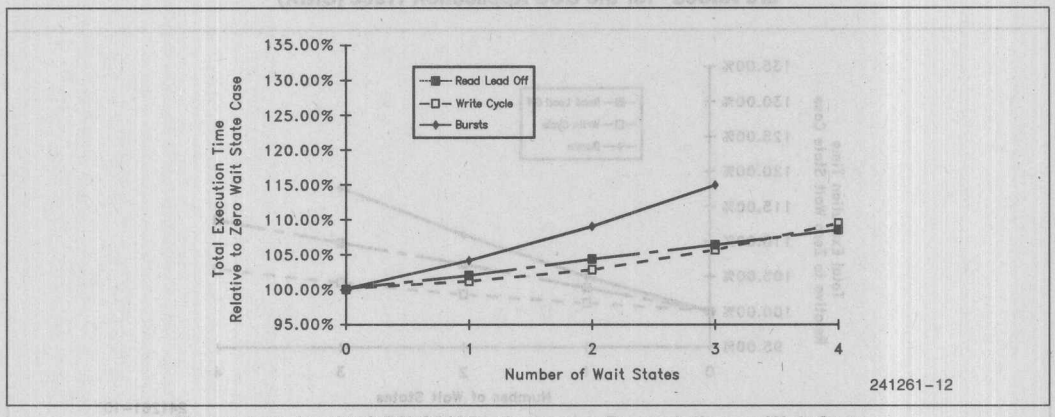


Figure 2.12. Intel486 DX CPU Performance Degradation as Wait States are Added - for the Pagemaker Application Trace (Windows)

Finally, the results for the Turbo C application under DOS are shown in Figures 2.13 and 2.14.

The rate of performance degradation for the Intel486 DX2 CPU with the Turbo C application is less than the UNIX and Windows examples. This is due to the lower external bus utilization of the application. However, the degradation is still about twice what it is for the Intel486 DX CPU. Note that the write importance is about equal to the burst importance in this case. This can be attributed to the greater percentage of writes in the bus cycle mix for this application.

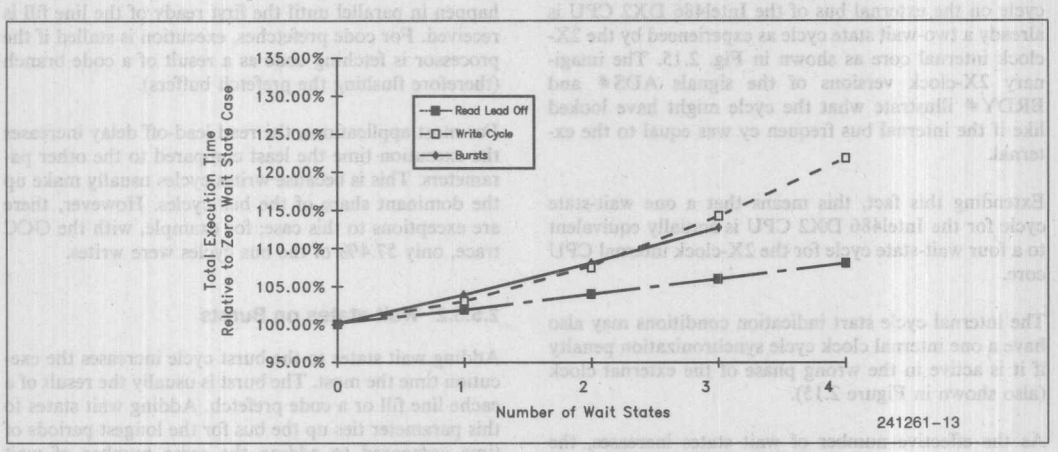


Figure 2.13. Intel486 DX2 CPU Performance Degradation as Wait States are Added - for the Turbo C Application Trace (DOS)

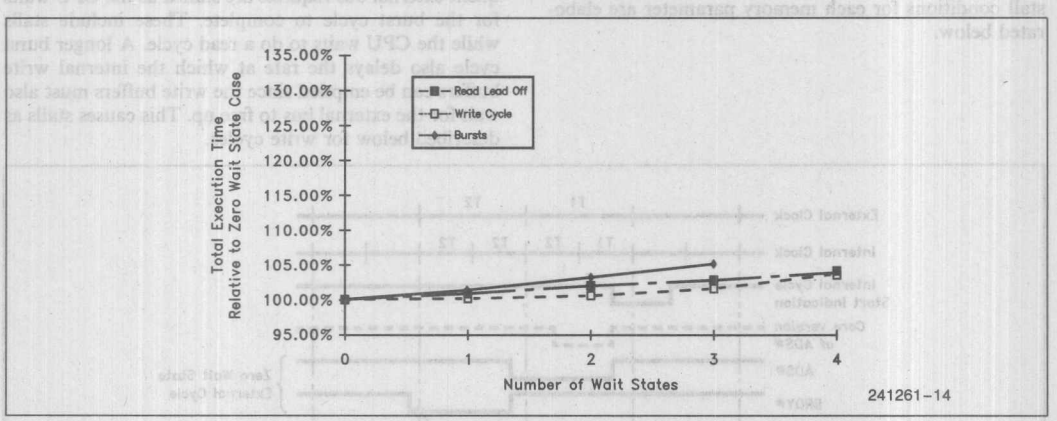


Figure 2.14. Intel486 DX CPU Performance Degradation as Wait States are Added - for the Turbo C Application Trace (DOS)

2.5.3 WAIT STATES AND CPU STALLS

The main reason why the relative performance of the Intel486 DX2 CPU degrades faster than the Intel486 DX CPU is that for every external wait state, two internal clock delays are caused. In fact, a zero wait state cycle on the external bus of the Intel486 DX2 CPU is already a two-wait state cycle as experienced by the 2X-clock internal core as shown in Fig. 2.15. The imaginary 2X-clock versions of the signals ADS# and BRDY# illustrate what the cycle might have looked like if the internal bus frequen cy was equal to the external.

Extending this fact, this means that a one wait-state cycle for the Intel486 DX2 CPU is actually equivalent to a four wait-state cycle for the 2X-clock internal CPU core.

The internal cycle start indication conditions may also have a one internal clock cycle synchronization penalty if it is active in the wrong phase of the external clock (also shown in Figure 2.15).

As the effective number of wait states increases, the CPU will stall program execution differently for each of the three memory parameters described above. The stall conditions for each memory parameter are elaborated below.

2.5.3.1 Delay Till First Ready of a Read

Wait states incurred on the first ready of a external read (the lead-off cycle) affect both data reads and code prefetches. For data reads, the CPU's execution is stalled under most conditions; no other operation can happen in parallel until the first ready of the line fill is received. For code prefetches, execution is stalled if the processor is fetching code as a result of a code branch (therefore flushing the prefetch buffers).

For most applications, the read lead-off delay increases the execution time the least compared to the other parameters. This is because writes cycles usually make up the dominant share of the bus cycles. However, there are exceptions to this case; for example, with the GCC trace, only 57.4% of the bus cycles were writes.

2.5.3.2 Wait states on Bursts

Adding wait states to the burst cycle increases the execution time the most. The burst is usually the result of a cache line fill or a code prefetch. Adding wait states to this parameter ties up the bus for the longest periods of time compared to adding the same number of wait states to the other parameters. As a result, all subsequent external bus requests are stalled as the CPU waits for the burst cycle to complete. These include stalls while the CPU waits to do a read cycle. A longer burst cycle also delays the rate at which the internal write buffers can be emptied since the write buffers must also wait for the external bus to free up. This causes stalls as described below for write cycles.

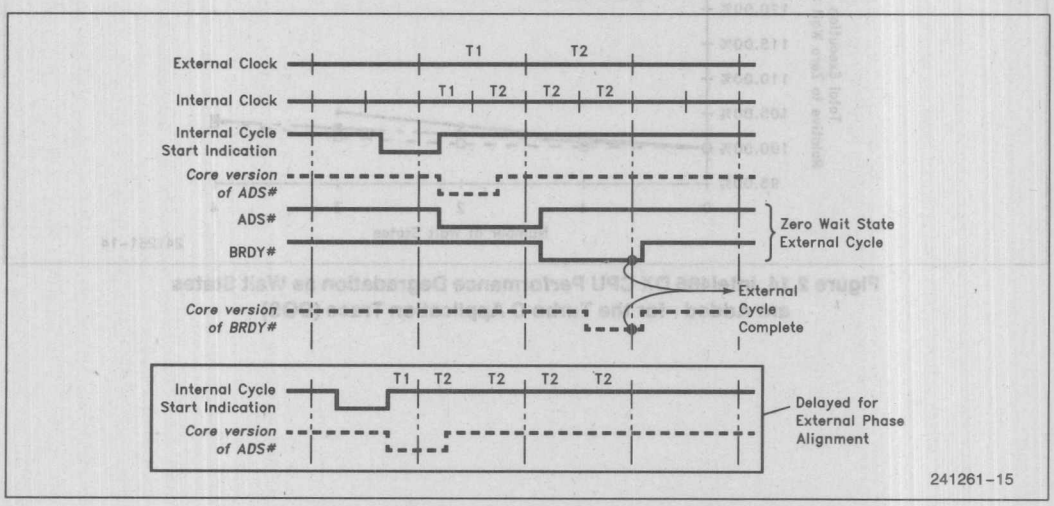


Figure 2.15. A Zero-Wait State Write for the Intel486™ DX2 CPU

Finally, longer code prefetch bursts will slow down CPU execution if the prefetch was a result of the prefetch queue being flushed. This is especially so if the instruction required extends beyond the first dword of the burst and therefore the CPU must wait for subsequent dwords before execution can start.

2.5.3.3 Write Wait States

There are three conditions under which a longer write cycle will stall CPU execution as additional wait states are added. These conditions are:

1. The write buffers are full and cannot accept any more writes.
2. A read cannot bypass the write buffers and must wait for them to be flushed.
3. A read bypasses the write buffers but must wait for an existing write cycle to complete.

Before these effects are elaborated, it is worthwhile to reexamine the operation of the internal write buffers.

The Intel486 DX2 CPU uses the same four-deep write buffers as the Intel486 DX CPU. The write buffers can accept data writes from the execution core as fast as one per clock. Once a write request is buffered, the internal unit that generated the request is free to continue processing. When all write buffers are full, any subsequent write transfer will stall inside the processor until a write buffer becomes available.

The bus interface unit can re-order pending reads in front of buffered writes. This is done because pending reads can prevent an internal unit from continuing, whereas buffered writes need not have a detrimental effect on processing speed. Writes are propagated to the external bus in the same first-in-first-out order in which they are received from the internal unit. However, a subsequently generated read request (data or instruction) may be reordered in front of buffered writes. As a protection against reading invalid data (reading stale data from a location in main memory when the location has been modified in the write buffers), this reordering of reads will only occur if all buffered writes are internal cache hits. Because an external read will only be generated for a cache miss, and will only be reordered in from of buffered writes if all such writes are internal cache hits, any read generated on the external bus will never read a location that is about to be written by a buffered write.

This reordering can only happen once for a given set of buffered writes, because the data returned by the read cycle could otherwise replace data about to be written from the write buffers.

The first condition that causes CPU stalls is when the write buffer is full. Write wait states decrease the rate at which the write buffer can be emptied. Since the Intel486 DX2 runs at a 2X-internal frequency, the likelihood of filling up the write buffers increases when compared to the Intel486 DX CPU as shown in Figure 2.16.

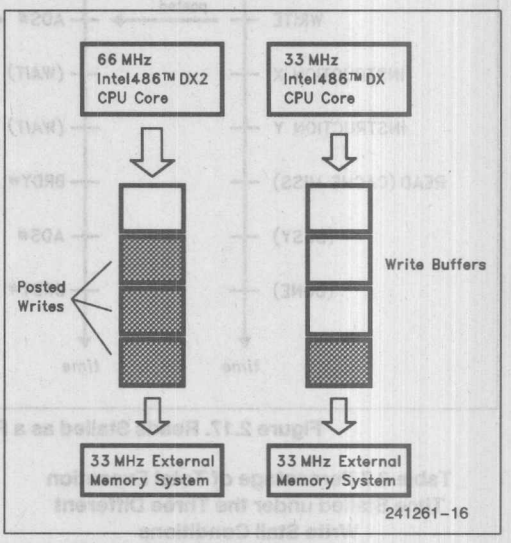


Figure 2.16. The Intel486 DX2 CPU's Write Buffers are More Heavily Used than the Intel486 DX CPU's

The second situation that degrades performance is during reads which cannot bypass the write buffers - either because the buffered writes were cache misses or because a read reordering had already occurred. These reads will be stalled until the write buffers are emptied. The more wait states required for writes on the external bus, the longer these stalls will last.

Finally, reads which can bypass the write buffers may be stalled by a write already in progress on the external bus. This condition is illustrated in Fig 2.17 for both the Intel486 DX and Intel486 DX2 CPUs. Note that for this example, although both the Intel486 DX2 and Intel486 DX CPUs take the same amount of time to complete the instruction stream, the Intel486 DX2 CPU is stalled longer (waiting for the write to complete) relative to its own internal 2X clock.

Out of the three conditions described above, stalls on write buffers full and stalls because of reads on busy writes dominate the increase in execution time as wait states are added to write cycles as shown in Table 2.7 for the SPEC1 trace. (The results shown in Figure 2.7 assume that the read and burst cycles complete in zero wait-states). These effects will be discussed again later when the addition of external write buffers is considered.

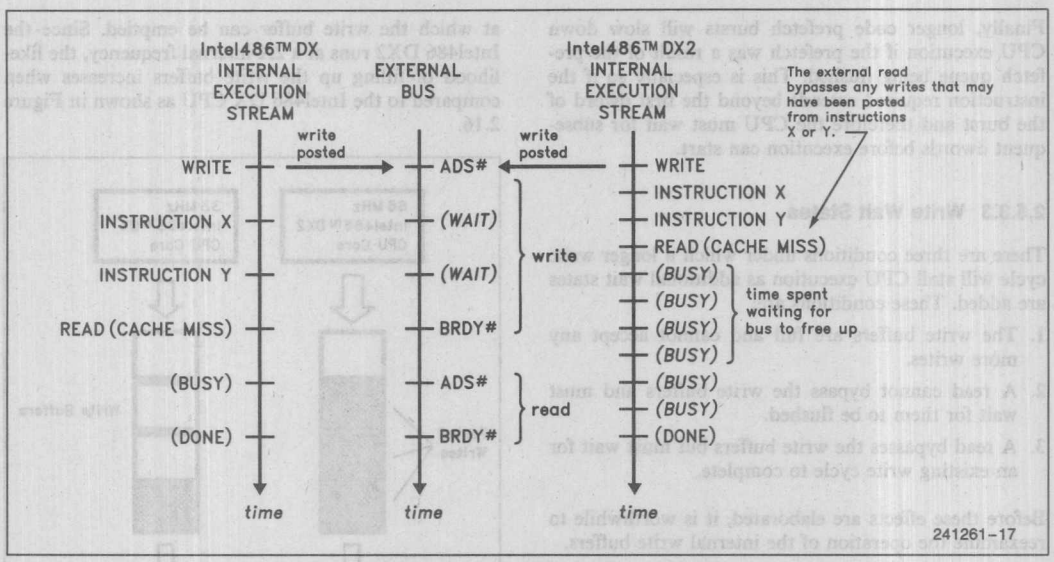


Figure 2.17. Reads Stalled as a Result of a Write Already In Progress

Table 2.7 Percentage of Total Execution Time Stalled under the Three Different Write Stall Conditions

	Intel486 DX CPU			Intel486 DX2 CPU		
Write wait states	**0**	**1**	**2**	**0**	**1**	**2**
Stalls on write buffers full	0.0%	0.1%	0.6%	1.0%	3.1%	6.5%
Reads cannot overtake writes	0.1%	0.1%	0.3%	0.3%	0.5%	0.5%
Stalls because of reads on busy writes	0.6%	1.3%	2.2%	2.4%	4.8%	7.5%

3.0 Memory Design Optimization

Some Intel486 DX2 CPU-based designs will include a memory system without an external cache. This section covers the design of such a cacheless memory system. Different memory architectures are discussed and the benefits of improving write performance through the addition of external write buffes will also be considered (see Figure 3.1).

Main memory performance will be important for external cache-based designs also, especially for applications with low external cache hit rates. It is recommended that the performance impacts of design choices in a cacheless memory design are understood even if you have already specified an external cache in your design.

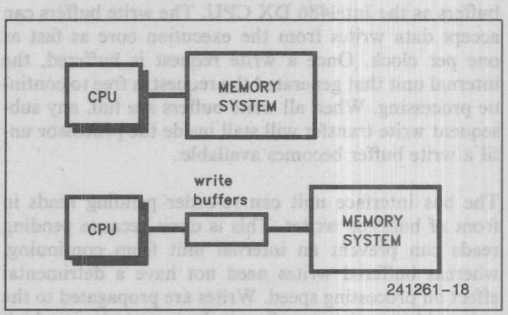


Figure 3.1 The Two Cacheless Intel486 DX2 CPU-Based Systems Considered

As discussed in the previous sections, the Intel486 DX2 Microprocessor requires a fast memory system for optimum performance. Memory systems that may have been adequate for Intel486 DX CPU designs running DOS applications may be suboptimal for the Intel486 DX2 CPU, especially running today's more demanding operating systems and applications.

Main memory page-mode operation and interleaving techniques are important for Intel486 DX2 CPU performance. These are commonly used in existing, well-designed Intel486 DX CPU memory systems. However, some systems still use non-interleaved memory designs borrowed from Intel386 DX systems. These will be less than optimal for a high performance Intel486 DX2 CPU workstations design.

3.1 Page Mode DRAM

Page-mode main memory controllers can be implemented in several fashions. Typical memory systems utilize paging for all accesses - during the beginning of a read, during read bursts and during write cycles; i.e. the RAS# line is held active after all accesses and only returned inactive during a page miss. Alternatively, paging may be used only for the burst portion of a read cycle; the RAS# line always returns inactive after the read or write cycle has been completed. This method is more commonly used in conjunction with a write-back external cache as discussed in Section 4.4.

For read burst cycles, the 16-byte linefill of data or code will always lie within a DRAM page, thereby allowing the data or code to be strobed out of memory with a series of back-to-back CAS# pulses. Paging allows for a much faster burst cycle compared to the case where a full RAS#-CAS# cycle is required for each dword. A page mode burst read access to a single bank of memory is shown in Fig. 3.2.

A paged memory system also allows for faster back-to-back write cycles. As was true for the Intel486 DX CPU, the Intel486 DX2 CPU generates writes in strings of two, about 60%-70% of the time, and writes in strings of three about 40%-50% of the time. This bus characteristic accounts for a large page hit rate for writes; therefore, it is faster to perform the back-to-back write cycles in a fast page mode rather than performing a full RAS#-CAS# cycle for each write.

At this point, it is worthwhile to examine the page hit/miss ratio for the three applications considered in the previous section. These results are shown in Table 3.1 and assume no external cache and a page size of 8192 bytes.

Table 3.1 Page Hit Ratios

	SPEC1	Pagemaker	Turbo C
Read Page Hit	31.2%	26.4%	25.4%
Read Page Miss	68.8%	73.2%	74.6%
Write Page Hit	65.5%	68.8%	68.9%
Write Page Miss	34.5%	31.2%	31.1%

Note the low page hit ratio for CPU reads. This is due to the internal cache of the Intel486 DX2 CPU that filters read requests and tends to make the cache miss reads more randomly distributed throughout main memory.

3.2 Interleaving

Interleaving involves the use of more than one bank of memory; different banks are controlled separately. As an access is occurring, the other banks are being readied for the next access. Interleaving can be implemented in several ways. Horizontally interleaved banks generate accesses for consecutive locations in memory. For example, one bank can be designated as an odd dword and another for the even dword. Vertically interleaved banks separate large contiguous regions of memory between banks; i.e. multiple DRAM pages are open. Memory controllers often combine both methods.

Horizontal interleaving can be combined with paging to generate very quick burst reads. Two 32-bit banks can generate a zero wait-state burst as detailed in the Intel Applications Note AP447 "A Memory Subsystem for the Intel486™ DX Family of Microprocessors including Second Level Cache." Fig. 3.3 illustrates a burst read cycle from a paged-interleaved memory system. The signals CAS0# and CAS1# drive each of the two 32-bit banks of memory in this example.

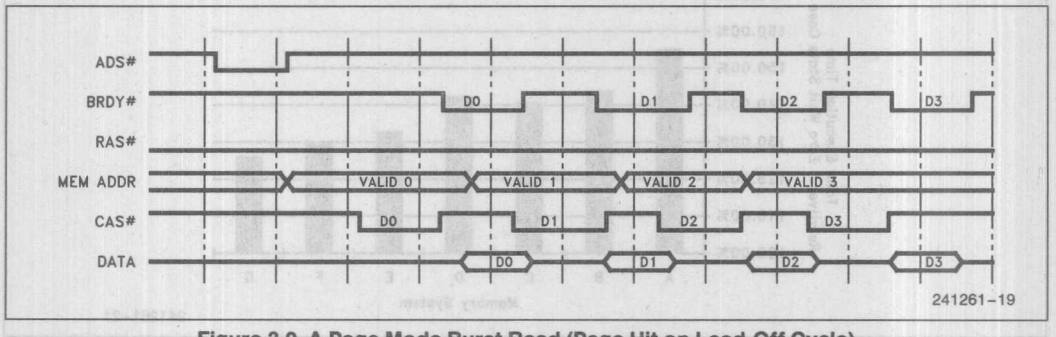


Figure 3.2. A Page Mode Burst Read (Page Hit on Lead-Off Cycle)

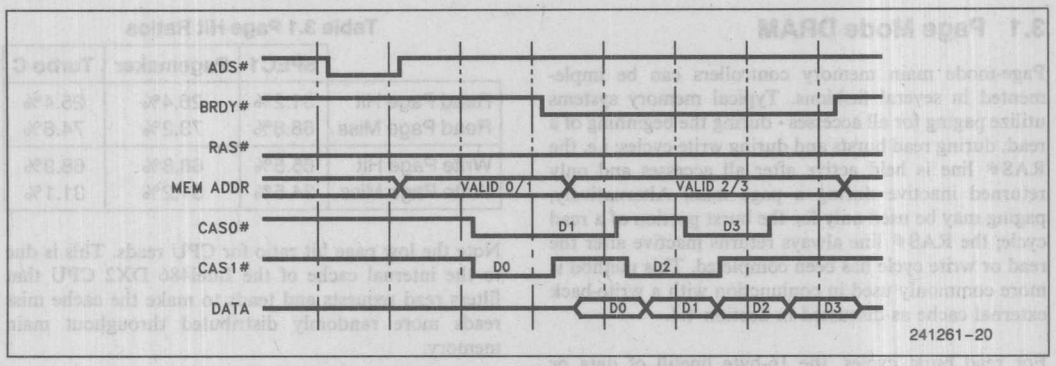


Figure 3.3. Burst Read Cycle from a Paged-Interleaved Memory System

Some implementations of a two-bank interleaved memory may be limited to a 1-2-1 burst cycle for the last three dwords. This is mainly limited by the amount of time it takes to invert the A3 address line between the second and third dwords.

3.3 Memory Read Performance Considerations

Seven memory systems with different read performance parameters are examined; write performance is kept constant during these simulations. The memory parameters are as follows:

Systems A through D represent the performance of some typical page mode memory controllers while the performance of systems E through G would require a paged-interleaved memory controller.

Table 3.2. Memory Systems used for the No-Cache System Test

	Read Page Hit	Read Page Miss	Write Page Hit	Write Page Miss
System A	4-3-3-3	8-3-3-3	3	6
System B	4-2-2-2	8-2-2-2	3	6
System C	4-2-2-2	7-2-2-2	3	6
System D	3-2-2-2	8-2-2-2	3	6
System E	3-1-2-1	7-1-2-1	3	6
System F	3-1-2-1	6-1-2-1	3	6
System G	3-1-1-1	6-1-1-1	3	6

The results for the Intel486 DX2 CPU with these memory systems is shown in Figure 3.4 through Figure 3.6 for the SPEC1, Pagemaker and Turbo C traces used previously. The graphs show the total execution time relative to the ideal zero-wait state memory system.

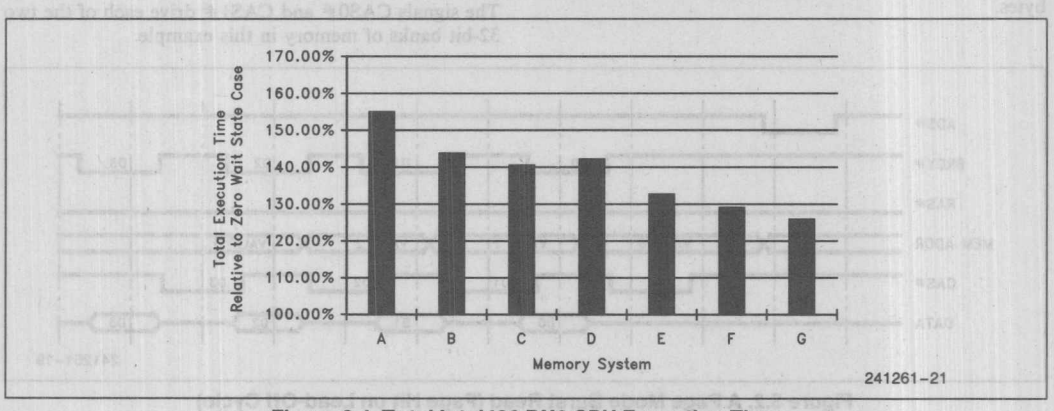


Figure 3.4. Total Intel486 DX2 CPU Execution Time versus Memory Read Performance - for SPEC1 (UNIX)

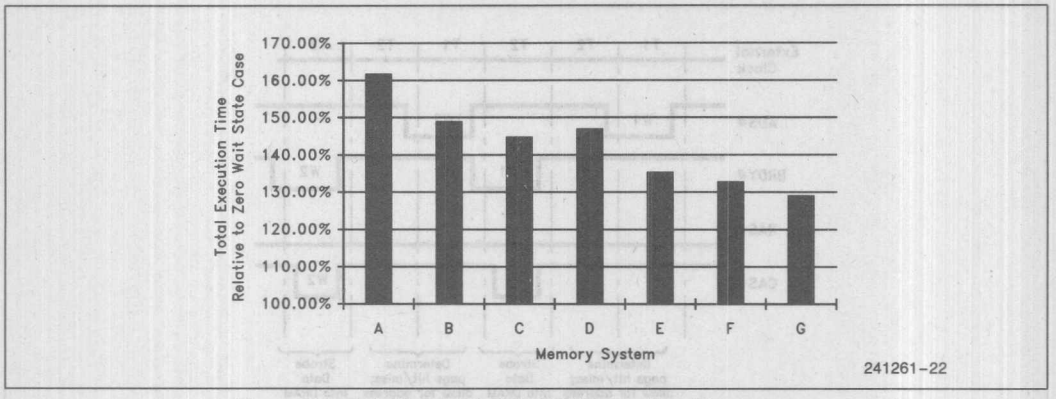


Figure 3.5. Total Intel486 DX2 CPU Execution Time versus Memory Read Performance - for Pagemaker (Windows)

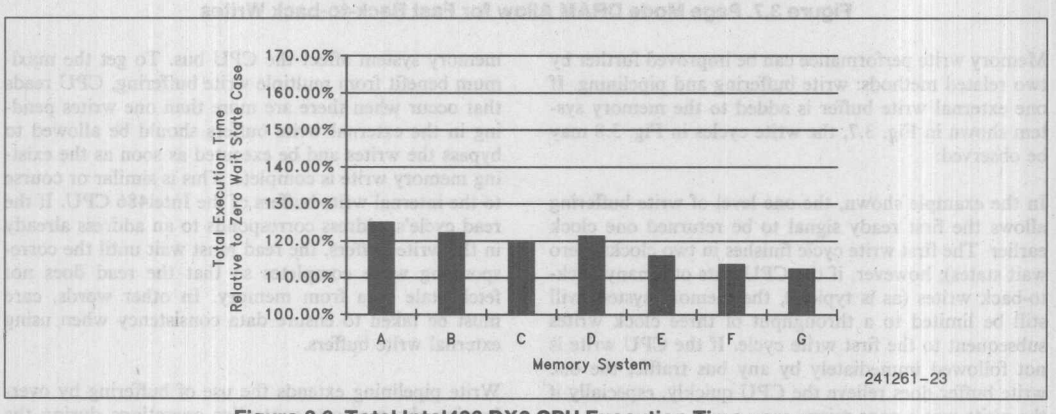


Figure 3.6. Total Intel486 DX2 CPU Execution Time versus Memory Read Performance - for Turbo C (DOS)

The interesting points to note here are:

- As expected, the DOS application suffers the least from slow memory performance.
- Burst performance is very important. Note the improvement from system A to B, system D to E and even from system F to G (where one clock was removed from the third burst).
- Since the read page hit ratio is lower than 50%, improving the read page miss lead-off cycle is more important than the read page hit lead-off cycle. Note the improvement from system B to C versus the improvement from system B to D.

3.4 Memory Write Performance Considerations

There are several methods of improving the write performance of the memory system. These methods are first described; the benefits of the different methods are discussed later.

The most common method for improving write performance is to employ page mode accesses to DRAM. As shown in Table 3.1, the page hit ratios for write cycles favor the use of page mode accesses. An example of a DRAM write cycle is shown in Fig. 3.7. On page hits, back-to-back three clock write cycles can be maintained. A page miss write would of course take an additional number of clocks to allow for the RAS# precharge time.

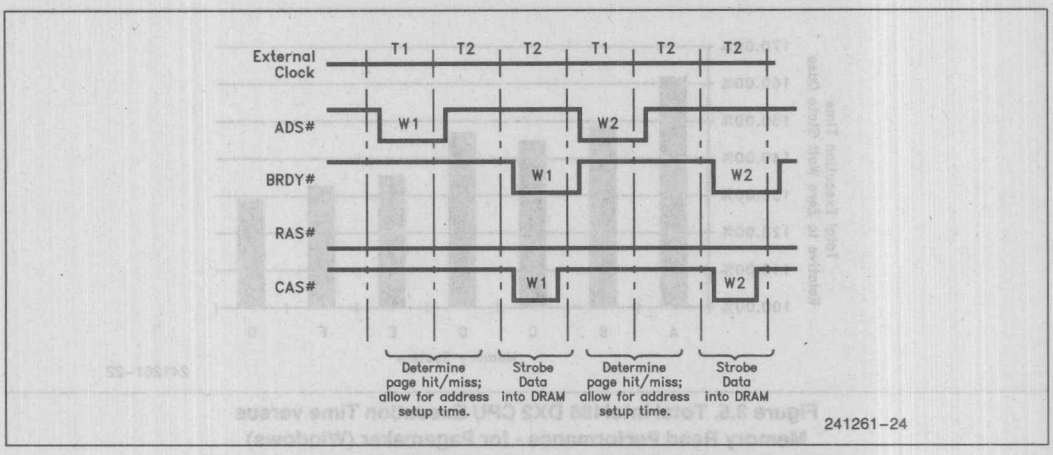


Figure 3.7. Page Mode DRAM Allow for Fast Back-to-back Writes

Memory write performance can be improved further by two related methods: write buffering and pipelining. If one external write buffer is added to the memory system shown in Fig. 3.7, the write cycles in Fig. 3.8 may be observed:

In the example shown, the one level of write buffering allows the first ready signal to be returned one clock earlier. The first write cycle finishes in two clocks (zero wait states); however, if the CPU puts out many back-to-back writes (as is typical), the memory system will still be limited to a throughput of three clock writes subsequent to the first write cycle. If the CPU write is not followed immediately by any bus traffic, the one write buffer does relieve the CPU quickly, especially if the write was a page miss.

More than one level of write buffering is sometimes employed. This would allow multiple writes to be accepted at zero wait states before wait states of the main memory system affect the CPU bus. To get the maximum benefit from multiple write buffering, CPU reads that occur when there are more than one writes pending in the external write buffers should be allowed to bypass the writes and be executed as soon as the existing memory write is complete. This is similar or course to the internal write buffers of the Intel486 CPU. If the read cycle's address corresponds to an address already in the write buffers, the read must wait until the corresponding write completes so that the read does not fetch stale data from memory. In other words, care must be taken to ensure data consistency when using external write buffers.

Write pipelining extends the use of buffering by overlapping the memory controller operations during the write cycle. An example is shown in Fig. 3.9 below. The data phase of the last write cycle (CAS# pulse) is overlapped in time with the address phase of the next memory write cycle (Page hit/miss decoding, etc.).

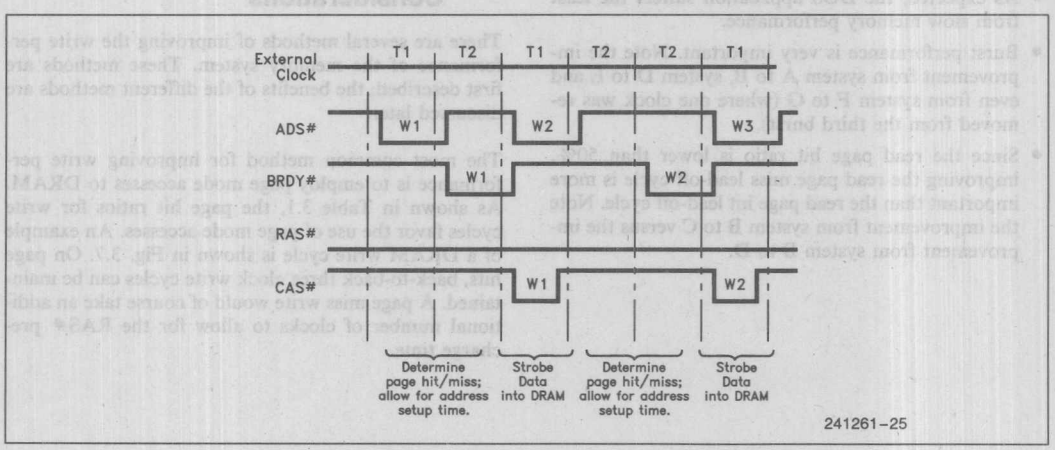


Figure 3.8. Adding One Write Buffer to the Memory System

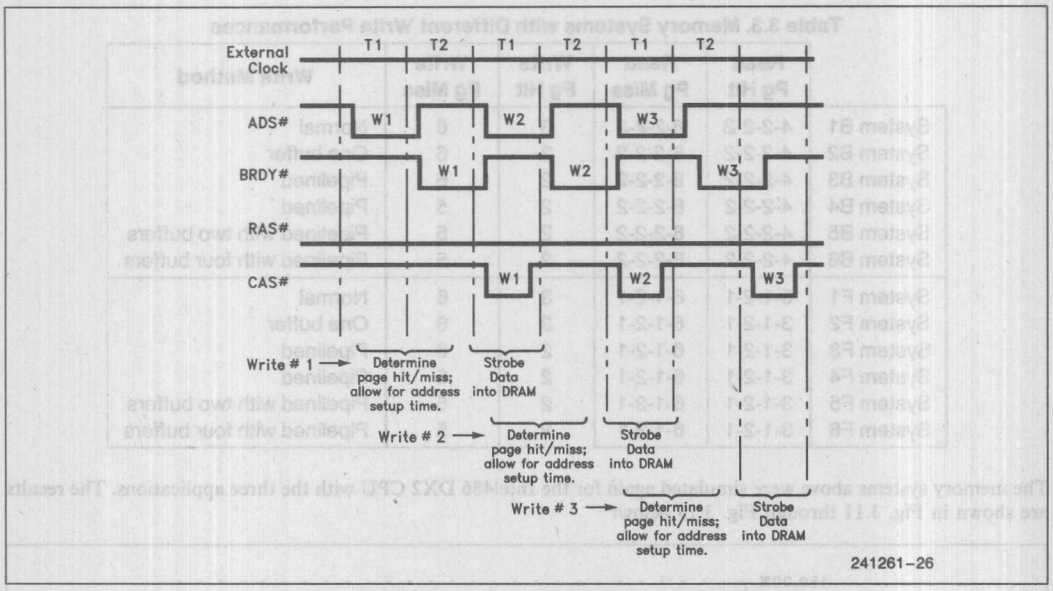


Figure 3.9. Pipelining the Writes to Memory

With pipelining, it is possible to achieve a throughput of many two-clock back-to-back page hit writes. (An example of write pipelining can be found in Intel Applications Note AP447) Note that pipelining may affect a subsequent read cycle; if the CPU read occurs immediately after the write, the beginning of the read will be delayed until the DRAM write cycle has completed. This is especially true for page miss writes where the write may take several clocks to complete. Note also that pipelined memory write systems can be combined with write buffering; this helps for the case where many back-to-back page miss writes occur.

Note that both write buffering and/or pipelining require a data path device between the CPU and main memory (see Fig. 3.10). This is needed to capture the data from the CPU before RDY# or BRDY# is returned, after which point the data will become invalid.

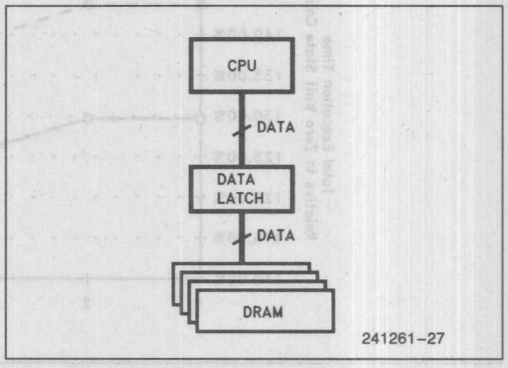


Figure 3.10. A Data Latch is Required for Write Buffering or Pipelining

To understand the performance benefits of the various methods described, systems B and F from the earlier simulations for read performance are repeated with different write performance parameters (refer to Table 3.3). Systems B and F were chosen as typical representations of paged and paged-interleaved memory controllers respectively.

Table 3.3. Memory Systems with Different Write Performances

	Read Pg Hit	Read Pg Miss	Write Pg Hit	Write Pg Miss	Write Method
System B1	4-2-2-2	8-2-2-2	3	6	Normal
System B2	4-2-2-2	8-2-2-2	3	6	One buffer
System B3	4-2-2-2	8-2-2-2	2	6	Pipelined
System B4	4-2-2-2	8-2-2-2	2	5	Pipelined
System B5	4-2-2-2	8-2-2-2	2	5	Pipelined with two buffers
System B6	4-2-2-2	8-2-2-2	2	5	Pipelined with four buffers
System F1	3-1-2-1	6-1-2-1	3	6	Normal
System F2	3-1-2-1	6-1-2-1	3	6	One buffer
System F3	3-1-2-1	6-1-2-1	2	6	Pipelined
System F4	3-1-2-1	6-1-2-1	2	5	Pipelined
System F5	3-1-2-1	6-1-2-1	2	5	Pipelined with two buffers
System F6	3-1-2-1	6-1-2-1	2	5	Pipelined with four buffers

The memory systems above were simulated again for the Intel486 DX2 CPU with the three applications. The results are shown in Fig. 3.11 through Fig. 3.13 below:

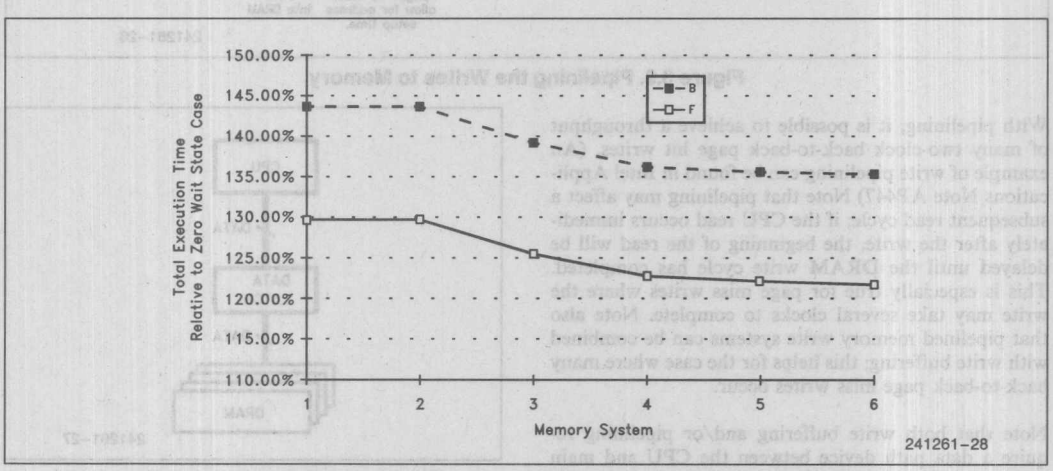


Figure 3.11. Total Execution Time versus Write Performance- for SPEC1 (UNIX)

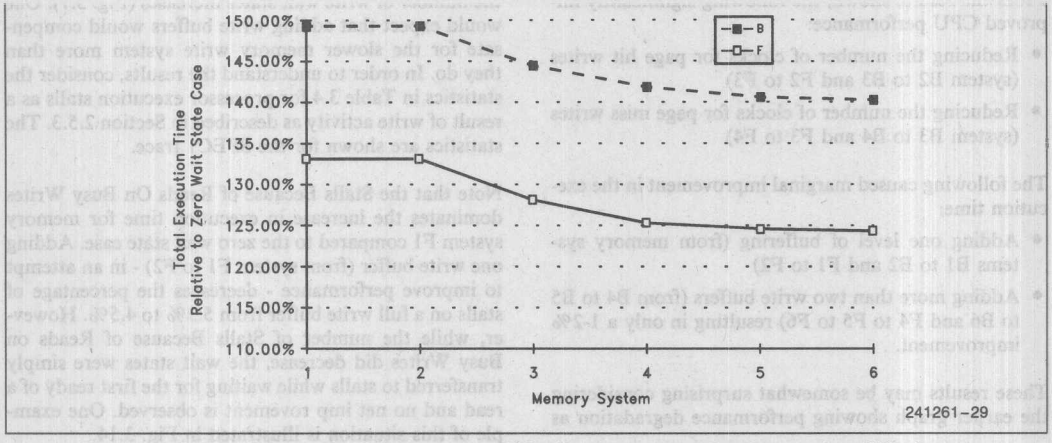


Figure 3.12. Total Execution Time versus Write Performance - for Pagemaker (Windows)

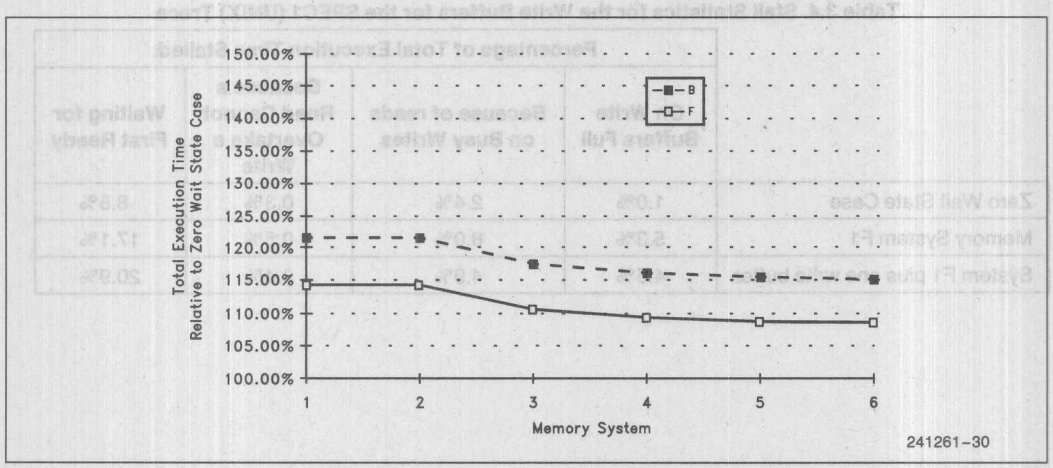


Figure 3.13. Total Execution Time versus Write Performance - for Turbo C (DOS)

From the results shown, the following significantly improved CPU performance:

- Reducing the number of clocks for page hit writes (system B2 to B3 and F2 to F3)
- Reducing the number of clocks for page miss writes (system B3 to B4 and F3 to F4)

The following caused marginal improvement in the execution time:

- Adding one level of buffering (from memory systems B1 to B2 and F1 to F2)
- Adding more than two write buffers (from B4 to B5 to B6 and F4 to F5 to F6) resulting in only a 1-2% improvement.

These results may be somewhat surprising considering the earlier graph showing performance degradation as the number of write wait states increases (Fig. 3.7). One would expect that adding write buffers would compensate for the slower memory write system more than they do. In order to understand the results, consider the statistics in Table 3.4 for processor execution stalls as a result of write activity as described in Section 2.5.3. The statistics are shown for the SPEC1 trace.

Note that the Stalls Because of Reads On Busy Writes dominates the increase in execution time for memory system F1 compared to the zero wait state case. Adding one write buffer (from system F1 to F2) - in an attempt to improve performance - decreases the percentage of stalls on a full write buffer from 5.3% to 4.5%. However, while the number of Stalls Because of Reads on Busy Writes did decrease, the wait states were simply transferred to stalls while waiting for the first ready of a read and no net imp rovement is observed. One example of this situation is illustrated in Fig. 3.14.

Table 3.4. Stall Statistics for the Write Buffers for the SPEC1 (UNIX) Trace

	Percentage of Total Execution Time Stalled:			
	On Write Buffers Full	Because of reads on Busy Writes	Because a Read Cannot Overtake a Write	Waiting for First Ready
Zero Wait State Case	1.0%	2.4%	0.3%	8.8%
Memory System F1	5.3%	8.0%	0.5%	17.1%
System F1 plus one write buffer	4.5%	4.9%	0.4%	20.9%

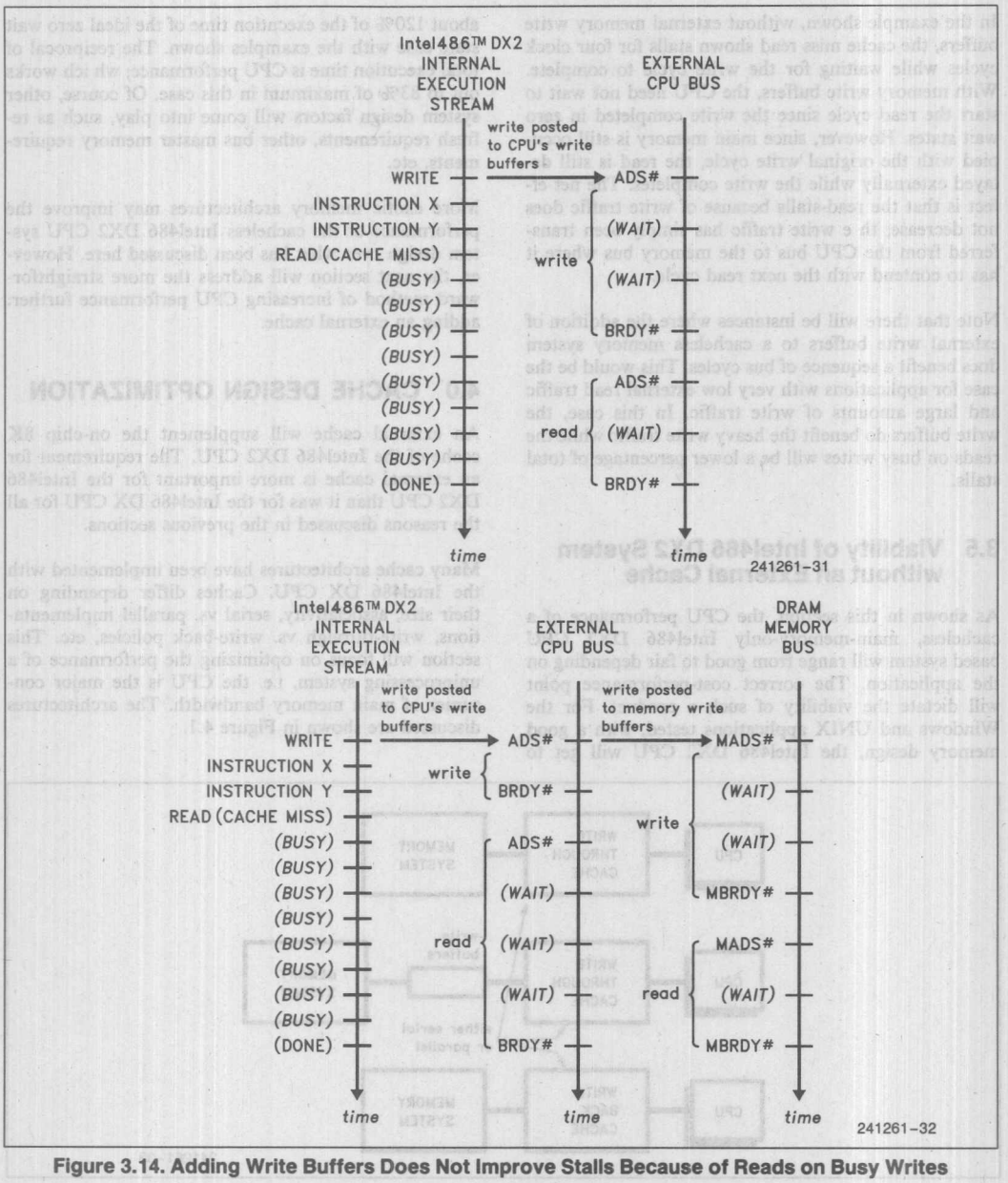


Figure 3.14. Adding Write Buffers Does Not Improve Stalls Because of Reads on Busy Writes

In the example shown, without external memory write buffers, the cache miss read shown stalls for four clock cycles while waiting for the write cycle to complete. With memory write buffers, the CPU need not wait to start the read cycle since the write completed in zero wait states. However, since main memory is still occupied with the original write cycle, the read is still delayed externally while the write completes. The net effect is that the read-stalls because of write traffic does not decrease; th e write traffic has simply been transferred from the CPU bus to the memory bus where it has to contend with the next read cycle.

Note that there will be instances where the addition of external write buffers to a cacheless memory system does benefit a sequence of bus cycles. This would be the case for applications with very low external read traffic and large amounts of write traffic. In this case, the write buffers do benefit the heavy write traffic while the reads on busy writes will be a lower percentage of total stalls.

3.5 Viability of Intel486 DX2 System without an External Cache

As shown in this section, the CPU performance of a cacheless, main-memory-only Intel486 DX2 CPU based system will range from good to fair depending on the application. The correct cost-performance point will dictate the viability of such a product. For the Windows and UNIX applications tested, with a good memory design, the Intel486 DX2 CPU will get to about 120% of the execution time of the ideal zero wait state case with the examples shown. The reciprocal of total execution time is CPU performance; wh ich works out to 83% of maximum in this case. Of course, other system design factors will come into play, such as refresh requirements, other bus master memory requirements, etc.

More exotic memory architectures may improve the performance of the cacheless Intel486 DX2 CPU system design over what has been discussed here. However, the next section will address the more straightforward method of increasing CPU performance further: adding an external cache.

4.0 CACHE DESIGN OPTIMIZATION

An external cache will supplement the on-chip 8K cache of the Intel486 DX2 CPU. The requirement for an external cache is more important for the Intel486 DX2 CPU than it was for the Intel486 DX CPU for all the reasons discussed in the previous sections.

Many cache architectures have been implemented with the Intel486 DX CPU. Caches differ depending on their size, associativity, serial vs. parallel implementations, write-through vs. write-back policies, etc. This section will focus on optimizing the performance of a uniprocessing system, i.e. the CPU is the major consumer of main memory bandwidth. The architectures discussed are shown in Figure 4.1.

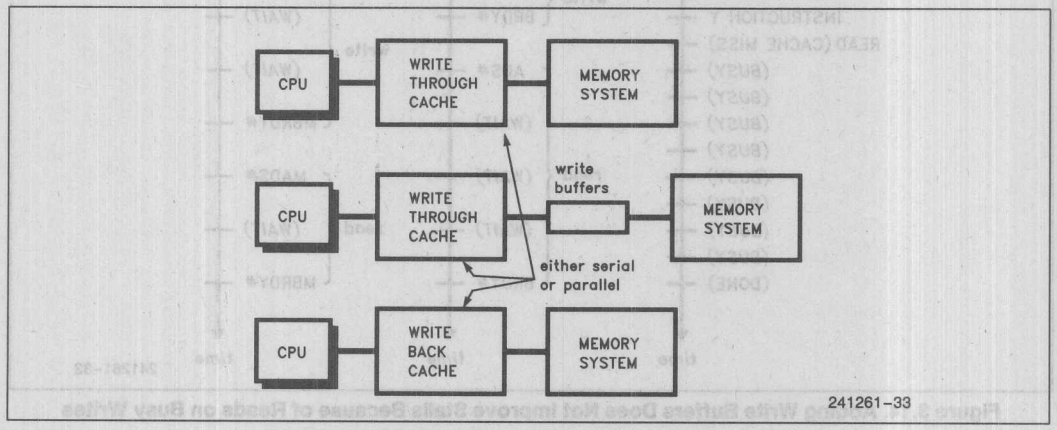


Figure 4.1. Different Cache Architectures Discussed

4.1 Overall Effect of an External Cache on CPU Performance

The same memory systems tested in the previous section are tested again with a 128K 2-way associative write-through parallel cache. This will yield the improvement achieved by the decrease in the effective number of read and burst wait states. The results are shown in Figures 4.2 through 4.4 for the three applications tested earlier.

The addition of an external write-through cache reduces the performance degradation caused by main-memory wait states for the lead-off cycle of a read and for wait states during the remainder of a burst. The impact of these wait states was discussed in Section 2.5.3.

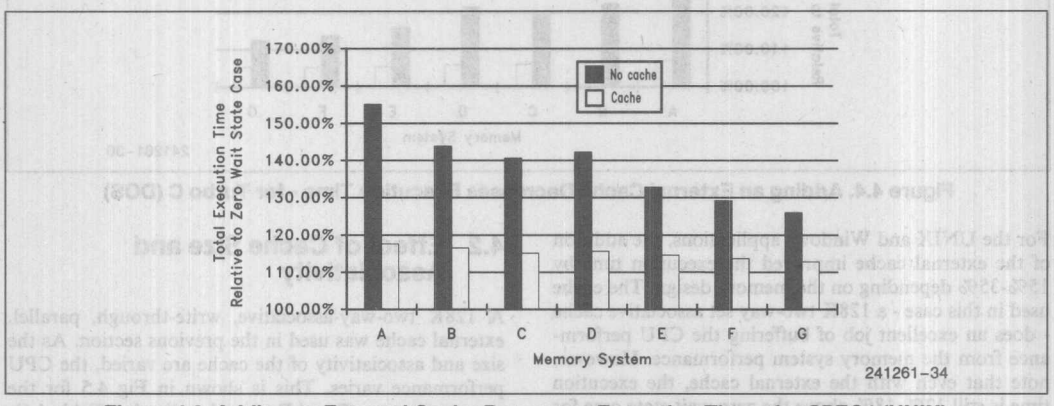


Figure 4.2. Adding an External Cache Decreases Execution Time - for SPEC1 (UNIX)

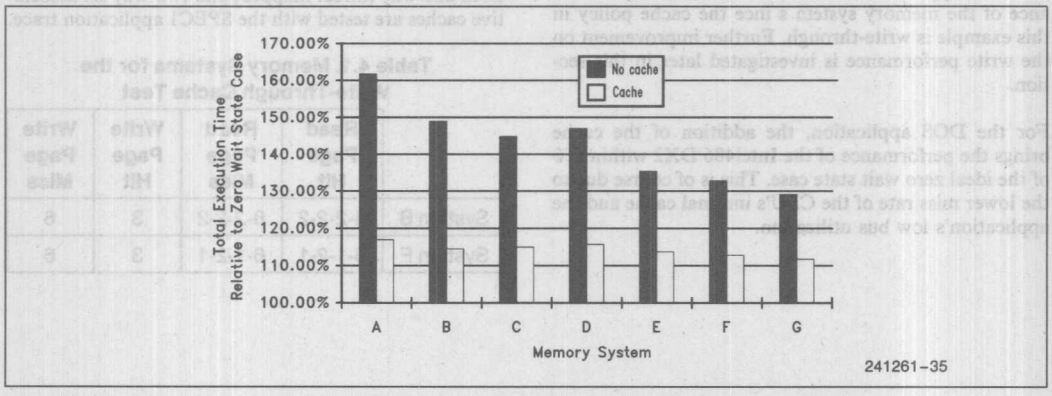


Figure 4.3. Adding an External Cache Decreases Execution Time - for Pagemaker (Windows)

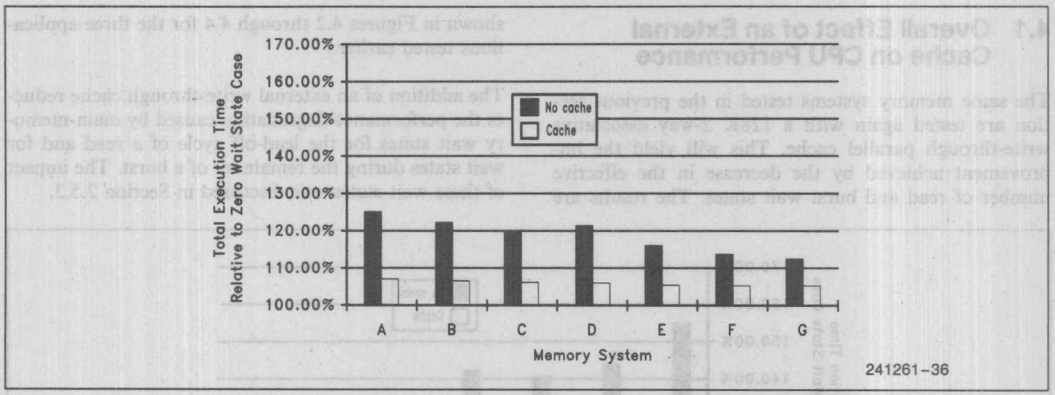


Figure 4.4. Adding an External Cache Decreases Execution Time - for Turbo C (DOS)

For the UNIX and Windows applications, the addition of the external cache improved the execution time by 15%-35% depending on the memory design. The cache used in this case - a 128K two-way set associative cache - does an excellent job of buffering the CPU performance from the memory system performance. However, note that even with the external cache, the execution time is still 12%-18% above the zero wait state case for these two applications. This is due to the write performance of the memory system s ince the cache policy in this example is write-through. Further improvement on the write performance is investigated later in this section.

For the DOS application, the addition of the cache brings the performance of the Intel486 DX2 within 5% of the ideal zero wait state case. This is of course due to the lower miss rate of the CPU's internal cache and the application's low bus utilization.

4.2 Effect of Cache Size and Associativity

A 128K two-way-associative, write-through, parallel, external cache was used in the previous section. As the size and associativity of the cache are varied, the CPU performance varies. This is shown in Fig 4.5 for the memory systems B and F as used earlier (see Table 4.1). Both one-way (direct mapped) and two-way set associative caches are tested with the SPEC1 application trace.

Table 4.1. Memory Systems for the Write-Through Cache Test

	Read Page Hit	Read Page Miss	Write Page Hit	Write Page Miss
System B	4-2-2-2	8-2-2-2	3	6
System F	3-1-2-1	6-1-2-1	3	6

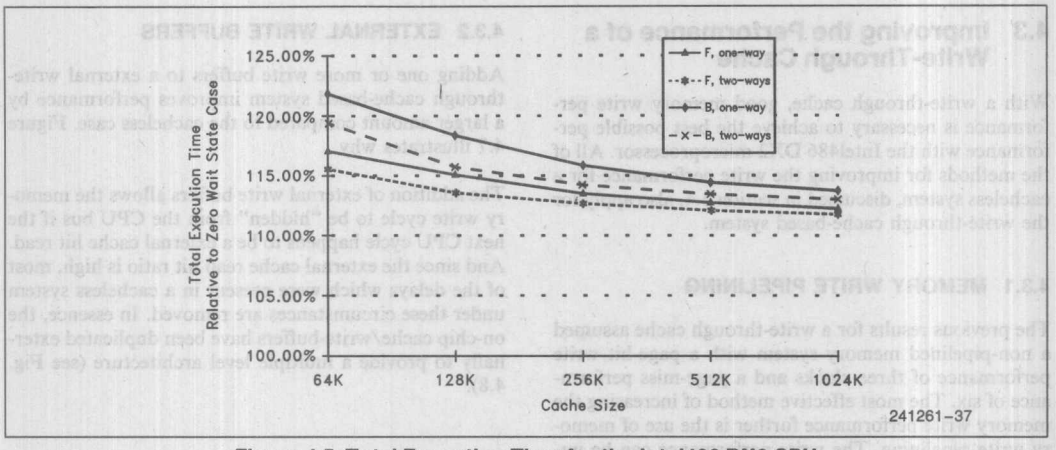


Figure 4.5. Total Execution Time for the Intel486 DX2 CPU as a Function of Cache Size and Associativity - for SPEC1

Fig 4.6 illustrates the external cache hit rates for CPU read cycles. The hit rates are directly related to the total execution time; higher hit rates result in shorter execution times.

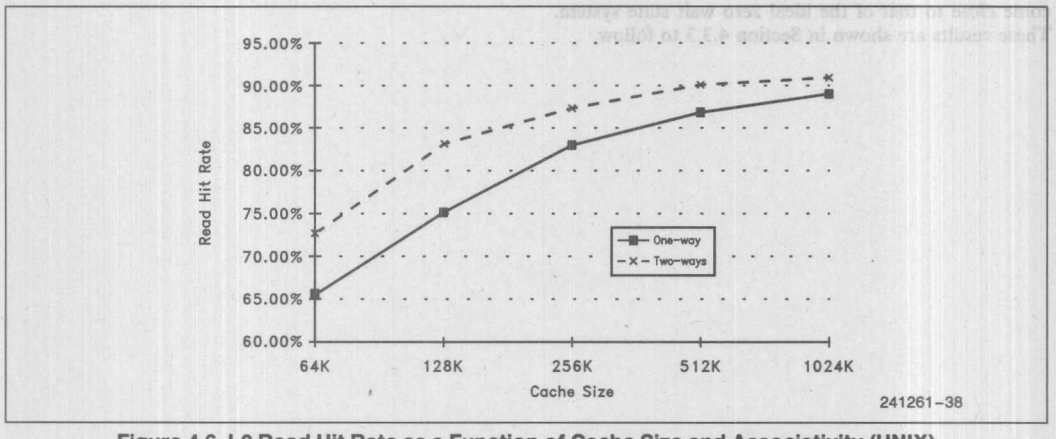


Figure 4.6. L2 Read Hit Rate as a Function of Cache Size and Associativity (UNIX)

4.3 Improving the Performance of a Write-Through Cache

With a write-through cache, good memory write performance is necessary to achieve the best possible performance with the Intel486 DX2 microprocessor. All of the methods for improving the write performance for a cacheless system, discussed in section 3.4, also apply for the write-through cache-based system.

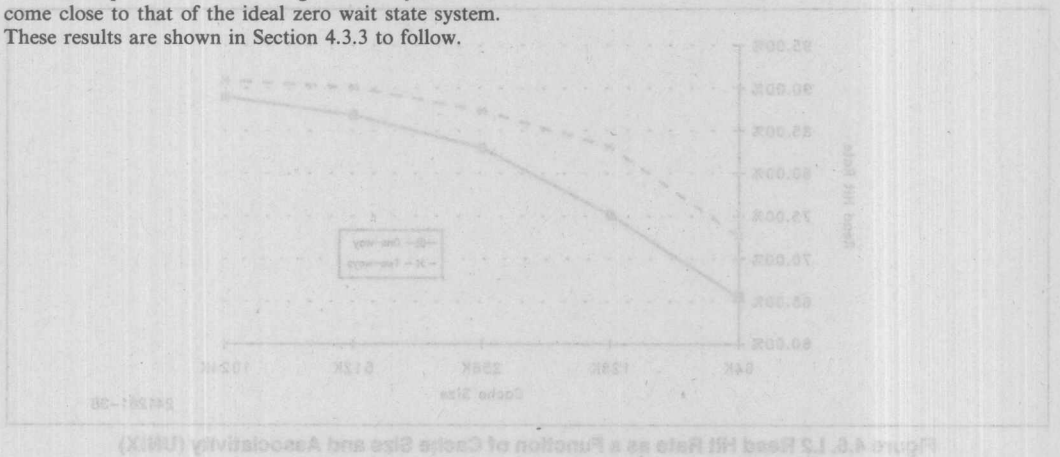
4.3.1 MEMORY WRITE PIPELINING

The previous results for a write-through cache assumed a non-pipelined memory system with a page-hit write performance of three clocks and a page-miss performance of six. The most effective method of increasing the memory write performance further is the use of memory write pipelining. The write performance can be improved so that continuous back-to-back page-hit write cycles can complete in zero wait-states. Pipelining can also reduce the number clocks required for a page-miss write cycle. As the write performance improves using this technique, the write-through cache system can come close to that of the ideal zero wait state system. These results are shown in Section 4.3.3 to follow.

4.3.2 EXTERNAL WRITE BUFFERS

Adding one or more write buffers to a external write-through cache-based system improves performance by a larger amount compared to the cacheless case. Figure 4.7 illustrates why.

The addition of external write buffers allows the memory write cycle to be "hidden" from the CPU bus if the next CPU cycle happens to be a external cache hit read. And since the external cache read hit ratio is high, most of the delays which were present in a cacheless system under these circumstances are removed. In essence, the on-chip cache/write-buffers have been duplicated externally to provide a multiple level architecture (see Fig. 4.8).



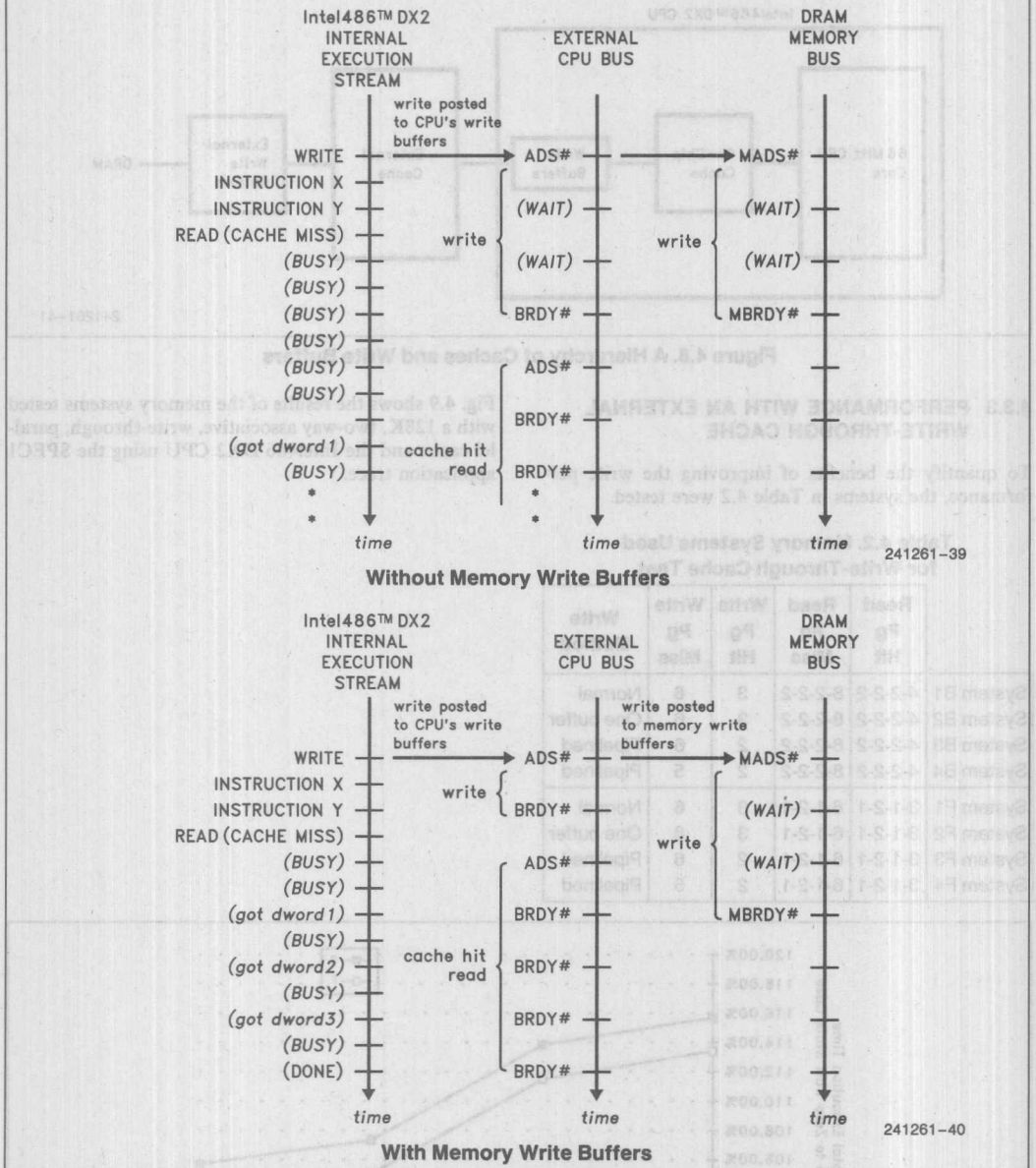


Figure 4.7. Adding External Write Buffers to an External Cache Reduces Execution Time

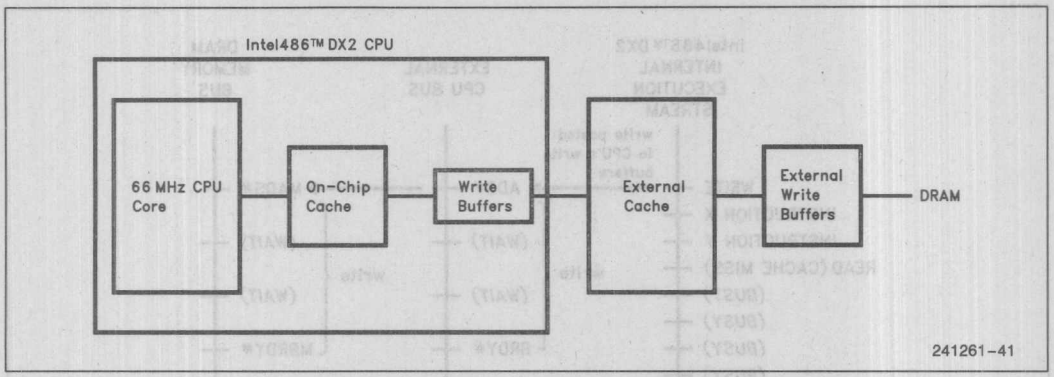


Figure 4.8. A Hierarchy of Caches and Write Buffers

4.3.3 PERFORMANCE WITH AN EXTERNAL WRITE-THROUGH CACHE

To quantify the benefits of improving the write performance, the systems in Table 4.2 were tested.

Table 4.2. Memory Systems Used for Write-Through Cache Test

	Read Pg Hit	Read Pg Miss	Write Pg Hit	Write Pg Miss	Write Method
System B1	4-2-2-2	8-2-2-2	3	6	Normal
System B2	4-2-2-2	8-2-2-2	3	6	One buffer
System B3	4-2-2-2	8-2-2-2	2	6	Pipelined
System B4	4-2-2-2	8-2-2-2	2	5	Pipelined
System F1	3-1-2-1	6-1-2-1	3	6	Normal
System F2	3-1-2-1	6-1-2-1	3	6	One buffer
System F3	3-1-2-1	6-1-2-1	2	6	Pipelined
System F4	3-1-2-1	6-1-2-1	2	5	Pipelined

Fig. 4.9 shows the results of the memory systems tested with a 128K, two-way associative, write-through, parallel cache and the Intel486 DX2 CPU using the SPEC1 application trace.

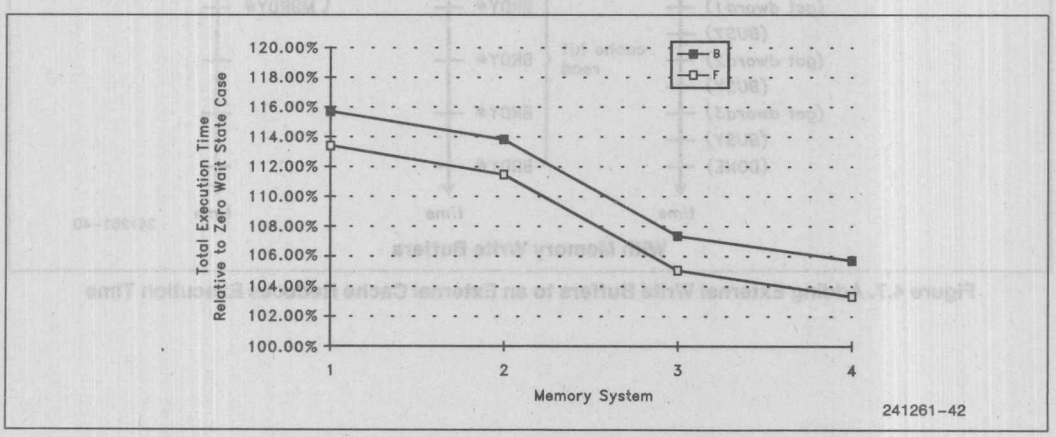


Figure 4.9. Improving the Write Performance Benefits a Write Through Cache - for SPEC1

As the write performance increases, the CPU performance approaches that of the zero wait state case. The improvement from systems B1 to B2 and from F1 to F2 illustrate the benefit of write buffering with an external cache. The improvement from systems B2 to B3 and from F2 to F3 reflect the benefit of memory write pipelining. Finally, the improvement from systems B3 to B4 and from F3 to F4 show how reducing the page-miss write performance also increases performance.

4.4 Write-Back Caches

If correctly implemented, a write-back external cache can provide good performance for a uniprocessing Intel486 DX2 CPU based system. Serial write-back caches have typically been used to reduce bus utilization for multiprocessing systems. The design complexity of a write-back cache controller is typically an order of magnitude higher than for a write-through cache controller. However, correct implementation is absolutely necessary if significant performance gains are to be realized with the Intel486 DX2 CPU.

A write-back cache is different from a write-through cache in that it allows cache write hits to modify the cache line without updating main memory. The cache has tags that include a bit called the modified (dirty) bit. This bit is set if the cache location has been written with new information and therefore contains information that is more recent than the corresponding information in main memory. If a subsequent read miss occurs and the line being fetched needs to fill the cache location that is currently being occupied by the modified line, the cache controller must then write the modified cache line back to main memory; hence coherency is maintained.

If a CPU write is not a cache hit, the cache controller has the option of allowing the write to propagate through to memory or to fetch the cache line from memory to be merged with the new write data. The cache line fill in the second option is called a write-allocation. In the following discussions, it is assumed that no write-allocations are being performed.

4.4.1 MAIN MEMORY CONTROLLER CONSIDERATIONS

The addition of an external write-back cache changes the characteristics of the main memory bus traffic. Since the cache effectively filters all CPU requests, the cycles that do propagate to main memory tend to be more distributed in their locations. This decrease in temporal and spatial locality will reduce the DRAM page hit rate as shown in Table 4.3 for a 128K, two-way associative, write-back cache with the SPEC1 application trace. Compare these results to the prior results in Table 2.1 for a cacheless system.

Table 4.3. Page Hit Ratios for a Write-Back Cache - for SPEC1

MEMORY CYCLES (100%)		SPEC1	PGMK	TURBOC
Reads:	Page Hits	17.1%	25.6%	24.7%
	Page Misses	13.8%	13.9%	12.9%
Writes:	Page Hits	58.9%	55.8%	40.8%
	Page Misses	10.2%	4.7%	21.6%

Therefore, it is less beneficial with a write-back cache to implement a page-mode main memory controller.

Of course, page mode DRAM accesses within the burst cycle are still important to retrieve the four words of a cache line quickly. This is also true for the write-back cycle where four dwords of the cache line must be written to memory. Memory controllers should be designed to support a burst write cycle instead of having to write each dword separately.

4.4.2 WRITE-BACK CYCLE

The write-back cycle is the sequence where a cache line fill from main memory has to displace a modified line that was already in the cache. The method in which the modified line is written back to main memory has an impact on overall CPU performance. Before analyzing the write-back cycle, consider first the architectures shown in Fig. 4.10.

In the simplest implementation, a write-back cache will share the data bus with the CPU and main memory as shown in configuration X. If this is the case, then during a write-back cycle, the modified line must be written back to main memory before the cache linefill can commence. This has a detrimental effect on performance since the CPU must wait while the write-back occurs. This sequence is shown in Figure 4.11.

With a data path device between the CPU-Cache bus and main memory as shown in configuration Y, the cache controller is able to defer the write-back of the modified data till after the linefill has completed. The CPU can continue execution after the linefill as long as subsequent cycles are all cache hits.

In configuration Z, a wider cache bus exists between the SRAM and the data path devices. This allows the modified data to be transferred more quickly from the SRAM to the data path device, thereby allowing the cache linefill to commence even sooner.

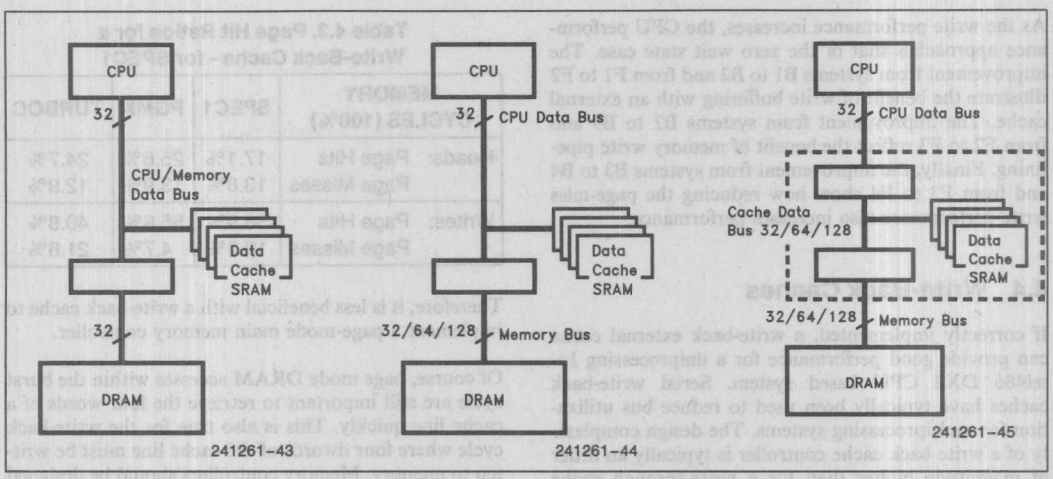


Figure 4.10. Different Architectures will Effect CPU Performance with a Write-Back Cache

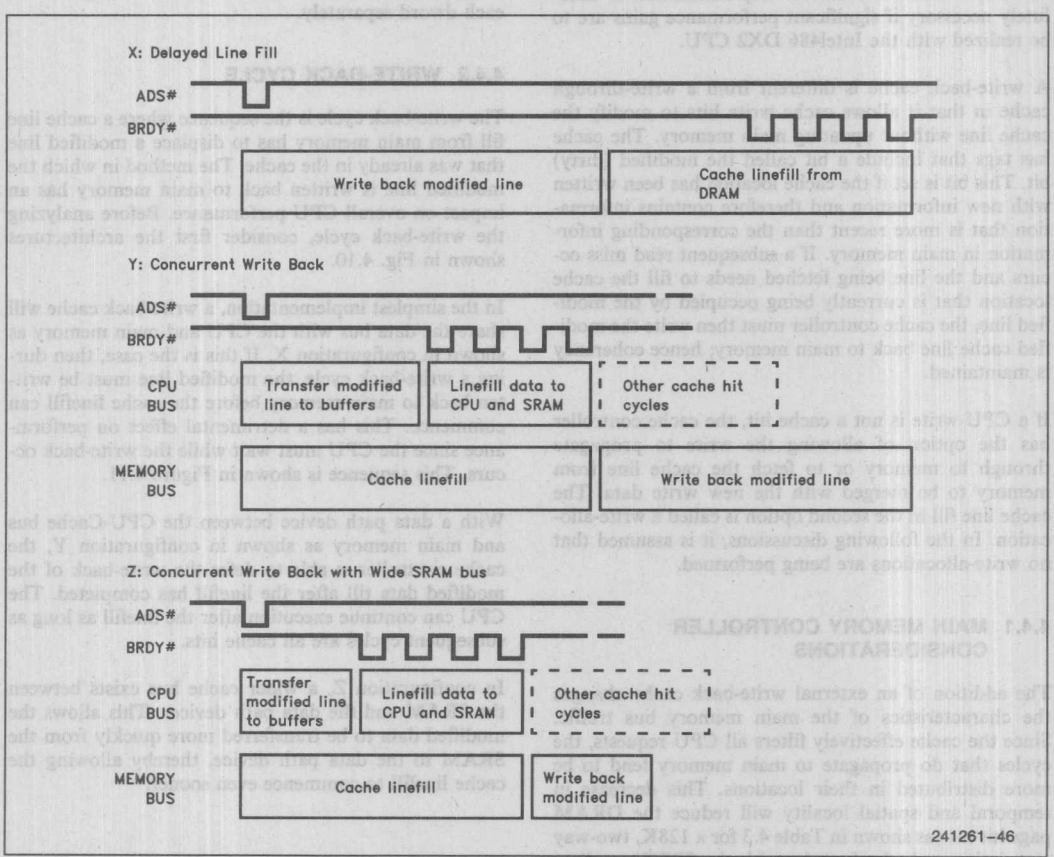


Figure 4.11. Different Implementations of the Write-Back Cycle

The following systems are used to demonstrate Intel486 DX2 microprocessor performance with different cache sizes and associativities.

The results are shown in Fig. 4.12 for the Intel486 DX2 CPU running the SPEC1 trace.

Table 4.4. Memory Systems used for Write-Back Cache Test

	Reads	Writes	Write-Back Method (described above)
System A	5-1-1-1	4-1-1-1 (burst)	Concurrent Write Back
System B	6-3-3-3	4-4-4-4 (non-burst)	Concurrent Write Back
System C	6-3-3-3	4-4-4-4 (non-burst)	Delayed Line Fill

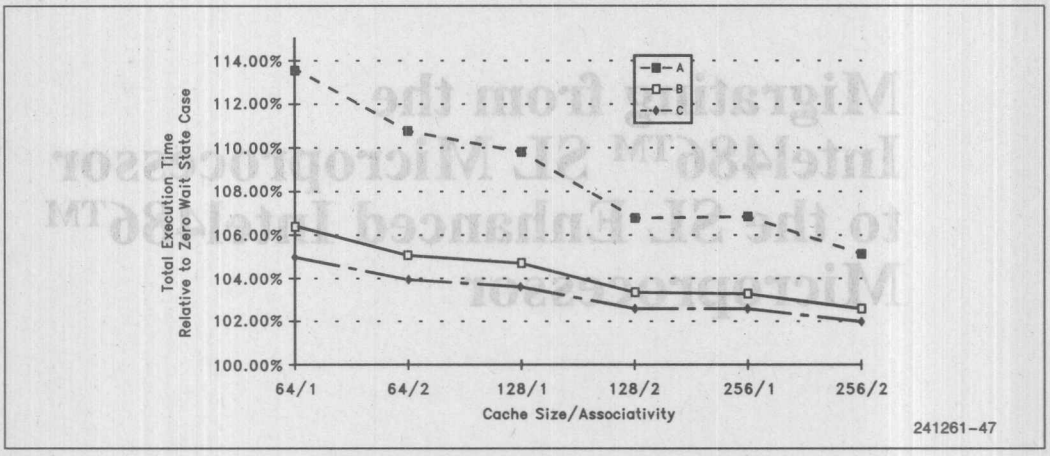


Figure 4.12. Intel486™ DX2 CPU Total Execution Time with Different Cache Size, Associativity, Memory Speed and Write-Back Method - for SPEC1

The addition of a write-back cache does an excellent job of decoupling the CPU performance from the main memory performance as shown with memory systems A and B. However, note that memory system B (with the delayed line fill) performs poorly - even worse than a good write-through cache - unless a significant amount of cache memory is added to reduce the miss rate.

5.0 CONCLUSION

This document has shown that good memory performance is especially important for the Intel486 DX2 microprocessor. Business workstation designs will require excellent CPU performance and will consequently have to incorporate well-designed, high-performance cache and memory systems.

In optimizing memory performance, an external cache is essential for hiding slow main memory access times. Write-through external caches offer good performance if coupled with good memory write performance. Write-back external caches can also offer excellent performance if designed correctly. Parallel write-back caches that cannot defer the write-back cycle till after a cache line fill will perform worse than a good write-through cache design.

intel®

AP-496

APPLICATION NOTE

Migrating from the Intel486™ SL Microprocessor to the SL Enhanced Intel486™ Microprocessor

DESMOND YUEN
MCG TECHNICAL MARKETING

October 1993

Order Number: 241810-001

Migrating from the Intel486™ SL Microprocessor to the SL Enhanced Intel486™ Microprocessor

CONTENTS

PAGE

INTRODUCTION 2-1044

1.0 COMPARISON OF THE SL ENHANCED INTEL486™ CPU AND INTEL486 SL CPU 2-1044

2.0 SYSTEM MANAGEMENT MODE IMPLEMENTATION 2-1045

2.1 System Management Interrupt .. 2-1045

2.1.1 General Design Considerations 2-1045

2.1.1.1 I/O Trapping 2-1045

2.1.1.2 Back-to-Back SMIs ... 2-1046

2.2. SMI Active (SMIACT#) 2-1046

2.2.1 General Design Considerations 2-1046

2.3 SMRAM Interfacing 2-1046

2.3.1 SMRAM Initialization 2-1047

2.3.2 General Design Considerations 2-1047

2.3.2.1 Accessing SMRAM ... 2-1047

2.3.2.2 Cache Coherency 2-1047

2.3.2.3 External Write Buffers 2-1048

2.3.2.4 A20M# Pin 2-1048

2.4 SMM Environment Initialization 2-1048

CONTENTS

PAGE

3.0 POWER MANAGEMENT 2-1049

3.1 STPCLK# Interrupt 2-1049

3.2 Global Standby Implementation 2-1050

3.2.1 Suspend Implementation .. 2-1050

3.2.1.1 Dynamic Clock Switching 2-1051

3.2.1.2 Power Consumption .. 2-1051

3.2.2 General Design Considerations 2-1051

4.0 RESET IMPLEMENTATION 2-1052

4.1 General Design Considerations 2-1053

CONCLUSION 2-1053

References 2-1053

2

INTRODUCTION

Since the introduction of the Intel386™ SL microprocessor and the subsequent introduction of the Intel486™ SL microprocessor, the SL Architecture has become the *de facto* standard for mobile computers. Given the industry acceptance of SL Architecture, Intel is extending the SL Architecture to the SL Enhanced Intel486 microprocessor family. This application note describes how the same features of the SL Architecture can be implemented on the SL Enhanced Intel486 CPUs. Although this application note is written for people with experience designing Intel486 SL CPU-based mobile computers, the information provided in this document will also be useful for anyone interested in learning more about the SL Enhanced Intel486 CPUs.

The first section of this document highlights the differences between the Intel486 SL CPU and the SL Enhanced Intel486 CPU. Section two describes the architectural differences in System Management Mode (SMM). Section three discusses power management features of the Intel486 SL CPU and the SL Enhanced Intel486 CPU. Section four explains reset implementation of the Intel486 SL CPU and the SL Enhanced Intel486 CPU.

1.0 COMPARISON OF THE SL ENHANCED Intel486™ CPU AND Intel486 SL CPU

The SL Enhanced Intel486 CPU supports many of the features available in the SL Architecture. The major difference between the SL Enhanced Intel486 CPU and the Intel486 SL CPU is level of integration. The Intel486 SL CPU is a highly integrated CPU with memory controller, ISA/PI-bus controller, and power management built into it. The SL Enhanced Intel486 CPU has retained all the SMM and power management features from the SL Architecture. Features not supported by the SL Enhanced Intel486 CPU can easily be implemented by external hardware. Table 1 highlights the differences between the SL Enhanced Intel486 CPU and the Intel486 SL CPU.

Table 1. Feature Comparison of the SL Enhanced Intel486 CPU and the Intel486 SL CPU

Features	SL Enhanced Intel486 CPU	Intel486 SL CPU
System Management Mode	Yes	Yes
SMBASE Relocation	Yes	No
Stop Clock	Yes	Yes
Upgrade Power-Down Mode	Yes	No
Package Options	168 lead PGA, 196 lead PQFP, 208 lead SQFP	196 lead PQFP, 208 lead SQFP
3.3V Operation	Yes	Yes
Clocking Options	1X clock input or 2X clock input	2X clock input
CPU Frequency	Intel486 SX CPU: 25 MHz, 33 MHz Intel486 DX CPU: 33 MHz, 50 MHz Intel486 DX2 CPU: 40 MHz, 50 MHz, 66 MHz	25 MHz, 33 MHz

2.0 SYSTEM MANAGEMENT MODE IMPLEMENTATION

System Management Mode (SMM), first introduced in the SL Architecture for notebook computers, provides a unique environment for software to perform power management functions much more efficiently. Since then, SMM has found its way into many new applications. The SMM hardware interface on the SL Enhanced Intel486 CPU is similar to that of the Intel486 SL CPU except for the handshaking protocol. The SL Enhanced CPUs handshake through the SMI and SMIACT# signals (see Figure 1), and the Intel486 SL CPUs handshake through the SMI and SMRAMCS# signals.

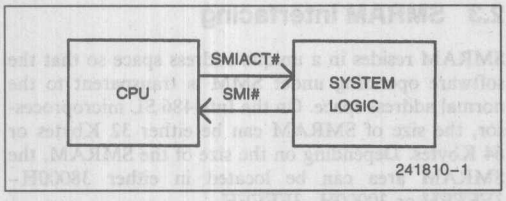


Figure 1. Basic SMI# Hardware Interface

2.1 System Management Interrupt

The system interrupts the normal program execution and invokes SMM by generating a System Management Interrupt (SMI#) to the CPU. On the Intel486 SL CPU, the SMI# input is held low as long as the CPU is in SMM. With the SL Enhanced Intel486 CPU, SMI# input only needs to remain active for a single clock provided the SMI setup and hold times, t20 and t21, are met. SMI# will also work correctly if it is held active for an arbitrary number of clocks.

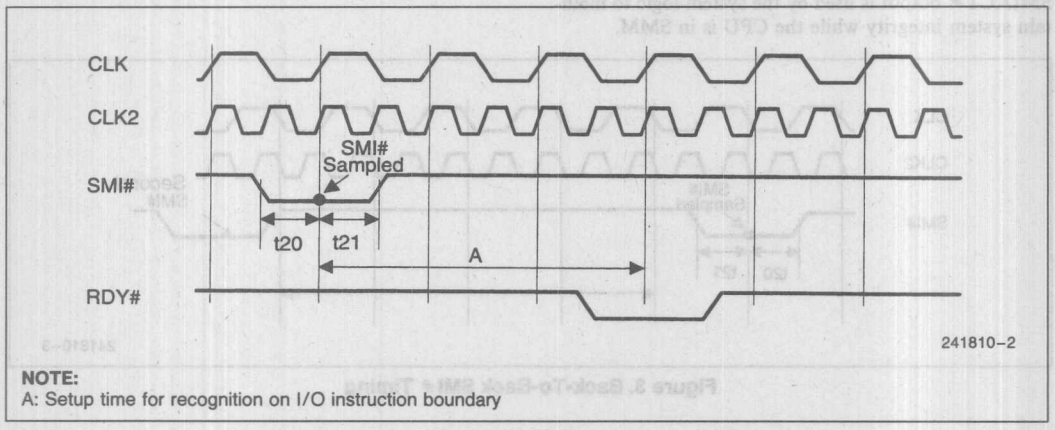
2.1.1 GENERAL DESIGN CONSIDERATIONS

For Intel486 SL processor-based systems, the 82360SL I/O generates the SMI request. For any SMI to be recognized by the SL Enhanced Intel486 CPU, the system logic must ensure all of the required timings are met. The following sections discuss the timing requirements that must be observed by the SMI generation logic interfacing to the SL Enhanced Intel486 CPU.

2.1.1.1 I/O Trapping

I/O trapping has proven to be very useful in the SL Architecture for device power management. Trapping the last I/O access prior to entering SMM prevents the CPU from accessing a powered-down device. With the exception of the SMFILO (System Management FILO), the SL Enhanced CPU supports the same I/O Instruction Restart option under SMM featured in the Intel486 SL CPU.

The I/O Instruction Restart feature of the SL Enhanced Intel486 CPU is used the same way as the I/O Instruction Restart feature of the Intel486 SL CPU. When the I/O Instruction Restart option is enabled (by setting offset 0FF00H in the SMRAM to 0FFH), the RSM instruction microcode modifies the restored EIP to point to the instruction immediately preceding the SMI# request, so that the I/O instruction can be re-executed. ***For the CPU to trap the last I/O access correctly, the external hardware must ensure the SMI# signal is asserted at least three CPU clock periods prior to asserting the RDY# signal (see Figure 2).***



NOTE:
A: Setup time for recognition on I/O instruction boundary

Figure 2. SMI# Timing when Servicing an I/O Trap

2.1.1.2 Back-to-Back SMIs

For back-to-back SMIs, the SMI# input must be held inactive for at least four clocks after it is de-asserted to reset the edge-triggered SMI detection logic. Otherwise, the second SMI# request may not be recognized (see Figure 3).

2.2 SMI Active (SMIACT#)

A new pin called SMIACT# (SMI ACTive) which indicates that the CPU is operating in SMM has been added to the SL Enhanced Intel486 CPU. The CPU asserts SMIACT# in response to an SMI interrupt request on the SMI# pin. SMIACT# is driven active by the CPU before accessing the SMRAM. SMIACT# remains active until the last access to SMRAM when the CPU restores (reads) its state from SMRAM. After the RSM instruction is executed, the CPU de-asserts the SMIACT# signal.

The SMIACT# signal is equivalent to the SMRAMCS# signal on the Intel486 SL microprocessor except that SMIACT# on the Intel486 SL CPU is active all the time and cannot be used as a chip select signal for external SMRAM. On the Intel486 SL microprocessor, the SMRAM is enabled automatically whenever the CPU is switched into SMM. A similar mechanism can also be implemented by using the SMIACT# signal. Whenever the SMIACT# signal is active, the SMRAM will be enabled by the system logic.

2.2.1 GENERAL DESIGN CONSIDERATIONS

As previously mentioned, one of the many uses of the SMIACT# output is to enable SMRAM when the CPU is operating in SMM. Most importantly, the SMIACT# output is used by the system logic to maintain system integrity while the CPU is in SMM.

If part of the system memory is overlaid by the SMRAM while the CPU is in SMM, the system logic should ensure that only the CPU and SMI handler have access to the SMRAM area. Accesses to addresses overlaid by a bus master or DMA controller when SMIACT# is active should be re-directed to the system memory underneath the SMRAM and not the SMRAM itself.

While inside SMM, the CPU should be protected from system activities such as CPU RESET, interrupt requests, and NMI, etc. The SMIACT# can be used by the system logic to block off these system activities while the CPU is in SMM.

2.3 SMRAM Interfacing

SMRAM resides in a unique address space so that the software operating under SMM is transparent to the normal address space. On the Intel486 SL microprocessor, the size of SMRAM can be either 32 Kbytes or 64 Kbytes. Depending on the size of the SMRAM, the SMRAM area can be located in either 38000H–3FFFFH or 30000H–3FFFFH.

On the SL Enhanced Intel486 CPU, the size of the SMRAM can be between 32 Kbytes and 4 Gbytes. The location of the SMRAM is determined by the SMBASE (SMRAM BASE ADDRESS) register, and defaults to SMBASE + 8000H, which is 38000H after CPU RESET. The first SMI after a CPU RESET always begins executing instructions at 38000H.

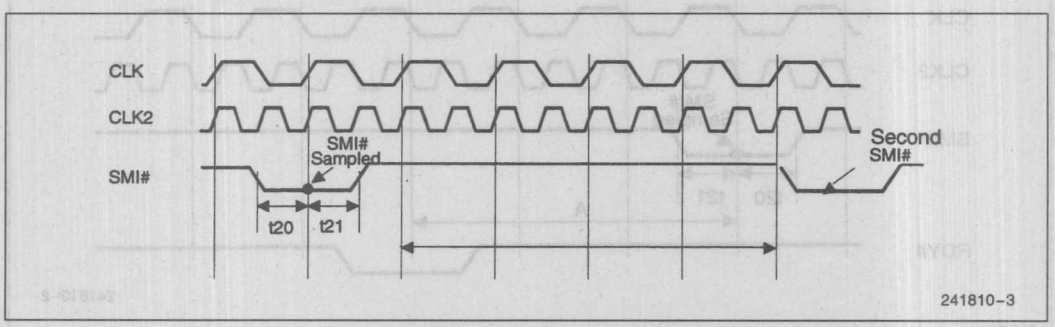


Figure 3. Back-To-Back SMI# Timing

2.3.1 SMRAM INITIALIZATION

The SL Enhanced Intel486 CPU family provides a new control register, SMBASE for changing the SMRAM base address (see Figure 4). The SMRAM base address can be changed after CPU RESET by invoking a dummy SMI call to change the SMBASE register.

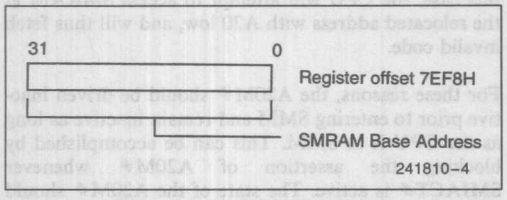


Figure 4. SMBASE Register

In the SL Enhanced Intel486 CPU, a new slot is added to the CPU dump area inside the SMRAM at offset 7EF8H for changing the SMRAM base address. During the execution of the RSM instruction, if the relocation bit is set, the CPU will read this slot and initialize the CPU to use the new SMBASE during the next SMI. From then on, the CPU will do its context save to the new SMRAM area pointed to by the SMBASE, store the current SMBASE in the SMM Base slot (offset 7EF8H), and then execute the new jump vector based on the current SMBASE.

SMBASE must start at a 32K aligned boundary. **Programming the SMBASE register to values that are not 32K aligned will cause the CPU to enter the shutdown state when executing the RSM instruction.** After the SMRAM base address is changed, the new starting address for the SMI jump vector is calculated by adding 8000H to the new SMRAM base address. The starting address for CPU state dump area will be remapped to the new SMRAM base address plus 0FFFFH.

A new bit (bit 17) has been added to the SMM Revision Identifier on the SL Enhanced Intel486 CPU to indicate whether the processor supports relocation of the SMRAM base address. With the SL Enhanced Intel486 CPU, the SMBASE relocation bit is always set to one to indicate the processor supports SMBASE relocation.

2.3.2 GENERAL DESIGN CONSIDERATIONS

Since the memory controller is embedded inside the Intel486 SL CPU, many design issues with SMRAM interface are handled internally by the CPU. For the SL Enhanced Intel486 CPU, these issues must be handled by the external system logic interfacing to the CPU.

2.3.2.1 Accessing SMRAM

Before the CPU can execute code inside SMM, the SMRAM must be loaded with valid SMM code. If the SMRAM is not initialized with code prior to entering SMM, executing invalid code out of the SMRAM can place the CPU in an unknown state. Thus, the external memory controller must provide a mechanism to bring the SMRAM into system address space without invoking SMM. This will allow software such as BIOS to load the SMM code into SMRAM.

The Intel486 SL CPU provides a hardware mechanism to access memory overlaid by SMRAM. Although system logic is not required to provide a mechanism to access memory located underneath SMRAM, it may be much easier to implement a suspend state (0-volt suspend or 5-volt suspend) if such a mechanism is provided.

2.3.2.2 Cache Coherency

Since the Intel486 SL CPU does not support a second level cache, cache coherency with SMRAM is handled completely inside the CPU. The CPU's internal cache is automatically emptied before entering SMM and after exiting SMM.

The SL Enhanced Intel486 CPU does not flush its cache before entering SMM or after leaving SMM. Cache flushing is not required if the SMRAM is located in a non-cacheable area in the memory address space or in an external address space which is not visible to the system. If the SMRAM is located in a cacheable area that overlays system memory, both the CPU internal cache and any second level caches must be flushed before entering SMM. If SMRAM is cacheable, the CPU internal cache and any second level caches must also be flushed when exiting SMM. The following steps must be taken by the system logic to maintain cache coherency when SMM overlays normal system memory:

1. Before entering SMM, the FLUSH# pin should be asserted when SMIACT# is driven active to empty the CPU cache.
2. The KEN# pin must be driven inactive to stop accesses to the SMRAM area from filling in the cache line if SMRAM is not cacheable.

3. Upon leaving SMM, if SMRAM is cacheable, the CPU cache is emptied by asserting the FLUSH# pin within one CPU CLK after the SMIACT# pin is de-asserted.

It is the responsibility of the system logic to ensure that the setup and hold times for FLUSH# and SMIACT# signals are met.

2.3.2.3 External Write Buffers

Like the Intel486 SL processor, the SL Enhanced Intel486 CPU empties its internal write buffers before entering SMM to prevent data in the write buffers from being written to SMRAM space. If a system supports a second level cache, the second level write buffers must also be emptied before the CPU enters SMM. It is possible that the CPU is in SMM before the second level write buffers are completely emptied by the memory controller. In case the second level write buffer is not completely emptied, the SMIACT# signal can be used to direct the memory write cycles to either SMM space or memory space.

2.3.2.4 A20M# Pin

The A20M# pin on the Intel486 CPU is provided to emulate the address wraparound at the 1 Mbyte boundary which occurs on the 8086 microprocessor (see Figure 5). The SMRAM space on the Intel486 SL CPU is always below 1 Mbyte memory address space. Memory above 1 Mbyte can either be accessed through the ISAWINDOW register or the MCWINDOW register. The A20M# signal is automatically driven low whenever the CPU is in SMM. When A20M# is active, all external bus cycles will drive A20 low, and all internal cache accesses will be performed with A20 low.

The SL Enhanced Intel486 CPU does not provide any memory mapping mechanism to access memory above 1 Mbyte. To access memory above 1 Mbyte inside SMM, the software has to disable the A20M# manually through the keyboard controller. Also, if the SMRAM is located above 1 Mbyte and A20M# is not enabled before entering SMM, the system will crash. In this case, the CPU will attempt to access SMRAM at the relocated address with A20 low, and will thus fetch invalid code.

For these reasons, the A20M# should be driven inactive prior to entering SMM and remain inactive as long as the CPU is in SMM. This can be accomplished by blocking the assertion of A20M# whenever SMIACT# is active. The state of the A20M# should be saved upon entry to SMM and restored to its original state after leaving SMM.

2.4 SMM Environment Initialization

When the CPU is running in SMM, the processor is in a pseudo "real mode" environment, but without the 64 Kbyte limit. After the SMRAM base address register has been relocated, the CPU segment registers will have values shown in Table 2 when an SMI# occurs. *The CS selector register still contains the value 3000H, not the value corresponding to the new SMBASE. The rest of the registers are still initialized to zero.*

If the SMRAM base address has not been relocated, the segment registers can be initialized in the same way as with the Intel486 SL processor, i.e., using the CS register which defaults to 3000H. Otherwise, the segment registers must be initialized correctly to point to the new SMRAM memory space.

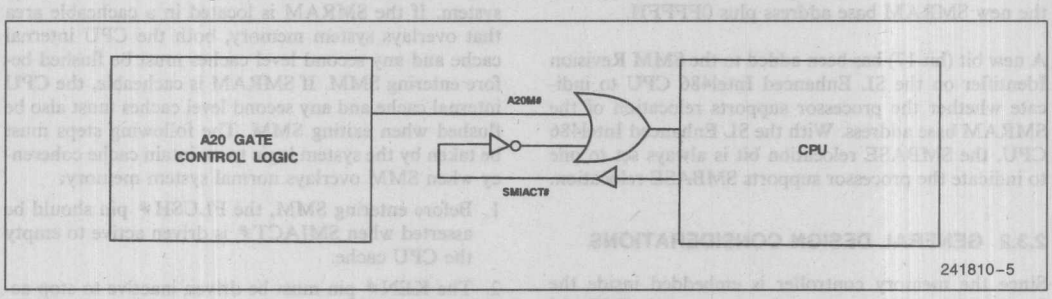


Figure 5. A20M# Interface Logic

Normally, the data segment registers are initialized to point to the SMRAM base address. Upon entering SMM, the CS BASE segment register is initialized to point to the SMRAM base address. The location of the SMRAM base address can be determined by reading the SMBASE register in the SMRAM at offset 0FEF8H *(the location of the SMRAM base address can also be stored in another memory location such as CMOS RAM by the BIOS which can be retrieved by the SMM program).*

The SMBASE contains a 32-bit address and has to be shifted to the right by four bits to generate a 16-bit segment address before it can be placed in the data segment selector registers. The CS selector register cannot be initialized by writing directly to it. It has to be initialized by executing a far jump instruction to an address within the SMRAM to force the CS selector register to point to the SMRAM base address.

When the CPU is in SMM, the operand size and the address size are still 16 bits but there are no limits to segment size. The physical address of an instruction is obtained by adding the value in CS segment base register to the value in EIP register, rather than the IP register. To access data anywhere within the four Gbyte logical address space, operand-size override (opcode 66H) and address-size override (opcode 67H) prefixes can be used as needed. Alternatively, SMRAM data located above 1 Mbyte can also be accessed by using 32-bit displacement registers.

Table 2. Register Values after SMI#

Segment Register	Selector	Base	Limit
CS	3000H	SMBASE	4 Gbytes
DS	0H	0H	4 Gbytes
ES	0H	0H	4 Gbytes
FS	0H	0H	4 Gbytes
GS	0H	0H	4 Gbytes
SS	0H	0H	4 Gbytes

3.0 POWER MANAGEMENT

One of the most important power management features on the Intel486 SL processor is CPU clock control. The clock control scheme on the SL Enhanced Intel486 CPU is similar to the Intel486 SL CPU, with the Intel486 SL CPU being driven by a 2X clock, and the SL Enhanced Intel486 CPU available with both the IX and 2X clocking options.

The 2X clock input is twice the internal frequency of the CPU, whereas the IX clock frequency is the same internal frequency of the CPU. With the IX clock, the two internal clock phases, "phase one" and "phase two", are generated by an internal Phase Lock Loop (PLL). The CPU clock input for a IX clock cannot be changed dynamically because the PLL requires a constant frequency CLK input (to within 0.1 %).

3.1 STPCLK# Interrupt

As with the Intel486 SL CPU, the SL Enhanced Intel486 CPU provides an interrupt mechanism, STPCLK#, which allows system hardware to control the power consumption of the CPU by stopping the internal clock to the CPU. Unlike the normal interrupts, INTR and NMI, the STPCLK# interrupt does not initiate interrupt acknowledge cycles or interrupt table reads.

The Stop Clock feature on the SL Enhanced Intel486 CPU has been improved, allowing the input to the STPCLK# to be driven asynchronously as well as synchronously. The major difference between asynchronous and synchronous control is that the STPCLK# interrupt latency is much shorter with asynchronous control.

With the Intel486 SL CPU, the STPCLK# input is controlled asynchronously through software. The STPCLK# is asserted after doing a dummy I/O read to the STPCLK register in the 82360SL or executing an HLT instruction. The STPCLK# signal will remain asserted until a system event wakes up the CPU. If the STPCLK# input is driven asynchronously, both setup and hold times t_{20} and t_{21} must be met for the STPCLK# interrupt request to be recognized.

After a STPCLK# interrupt request is recognized by the CPU, the processor will stop execution on the next instruction boundary, stop the pre-fetch unit, and then empty all internal pipelines and the write buffers. Finally, a special Stop Grant bus cycle is generated. The pin state during a Stop Grant cycle is shown in Table 3.

The interrupt acknowledge cycle is terminated when the system logic returns RDY# or BRDY#. At this point the CPU is in the Stop Grant state and the internal clock is stopped. The Stop Grant cycle is similar to the HALT cycle except that the address bus has the value 04H instead of 00H.

Table 3. Pin State During Stop Grant Cycle

Signals	State
M/IO#	0
D/C#	0
W/R#	1
Address Bus	0000 0010H (A_4 = 1)
BE3#–BE0#	1011 (same as HALT)
Data bus	Floated

Using the STPCLK# input, the SL Enhanced Intel486 CPU can be put into low power states similar to the Global Standby and Suspend states as with the Intel486 SL microprocessor.

3.2 Global Standby Implementation

In an Intel486 SL processor-based system, the 82360SL puts the CPU in a low power standby state (CPU I_{CC} ~ 20 mA–50 mA) when the system is in Global Standby. A similar state called **Stop Grant State** is provided by the SL Enhanced Intel486 CPU. The Stop Grant state can be entered by simply asserting the external STPCLK# interrupt pin. Once the STPCLK# interrupt is acknowledged by the CPU (i.e., after the Stop Grant cycle is placed on the bus), the CPU is in the Stop Grant state.

While in the Stop Grant state, the CPU still responds to RESET or SRESET and requests a cache invalidation (i.e., HOLD, AHOLD, BOFF# and EADS#). However, the CPU does not recognize any other inputs while in the Stop Grant state. Input signals to the CPU will not be recognized until one CPU clock cycle after STPCLK# is deasserted.

To emulate Global Standby, the stop clock control logic must be able to de-assert the STPCLK# signal whenever there is system activity (i.e., INTR, IRQ, NMI, and SMI#). Typically, the stop clock de-assertion logic is implemented by logic which latches the incoming interrupt requests from the system. The CPU returns to its normal state within 10–20 CPU clock cycles after exiting the Stop Grant state.

As mentioned before, the CPU does not recognize any interrupt request while the STPCLK# input is active. To prevent the interrupt request from getting lost, the interrupt request logic must ensure the interrupt signal is held active for at least one CPU clock after the STPCLK# input is de-asserted.

3.2.1 SUSPEND IMPLEMENTATION

From the Stop Grant state, the CPU can go into a lower power state similar to the suspend state offered in the Intel486 SL processor. After the CPU is in the Stop Grant state, the CPU can enter the lowest power **Stop Clock state (~ 100 μA–200 μA)** by stopping the CPU clock input. The CPU clock input can be driven to either logic high or logic low during Stop Clock state. The CPU will not generate any acknowledge cycle when entering stop clock state.

For a 2X clock input, the clock input to CLK2 can be stopped on either a logic high or logic low independent of the clock phase. The CPU will go back to Stop Grant state as soon as the CPU clock is re-started. Upon exit from Stop Clock State, the CPU clock input must be re-started in the same state when it was stopped (see Figure 6).

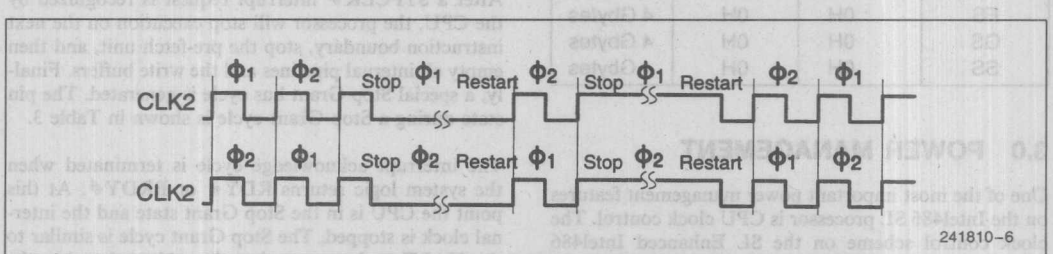


Figure 6. CLK2 Phase Coherence in CLK2 Stop and Restart

For a CPU with a 1X clock input, the CPU clock can be stopped in the same manner as a CPU with a 2X clock input. Because of the phase lock loop, the CPU will not return to the Stop Grant state right after the CPU clock input has been re-started. To allow time for the PLL to stabilize, the CPU clock input must be held at a constant frequency for a period of time equal to the PLL startup latency (as specified in the data book) before the CPU will return to the Stop Grant state.

As long as the CPU clock input is stopped, the system logic must keep all the CPU input signals in the same state before the clock was stopped. Any change in state on an input signal (except for INTR) before the CPU has returned to the Stop Grant state will result in unpredictable behavior. The CPU will not be able to recognize any interrupt request while the CPU clock is stopped.

3.2.1.1 Dynamic Clock Switching

For a CPU clock with a 1X clock input, the CPU clock cannot be changed "on-the-fly". For power management as well as implementation of features such as deturbo mode, it is advantageous to run the CPU clock at a lower frequency. This can be accomplished by putting the CPU into Stop Clock state and change the CPU clock to a lower speed. After the CPU clock input frequency is changed, the clock control logic must ensure that the clock input has been running at a constant frequency for the time period necessary for the PLL to stabilize before de-asserting the STPCLK# signal. The lowest CPU clock rate for a 1X part is 8 MHz (see Figure 7).

3.2.1.2 Power Consumption

The Stop Grant and Stop Clock states are designed to save power. While the processor is in Stop Grant state, the input/output signals on the CPU remain at the same state when entering the Stop Grant state, floated (data and parity signals), or driven to a different state. If some of the signals are driven improperly, the system can end up consuming more power.

To achieve the lowest power consumption, all the possible current leakage must be eliminated. The system logic should never drive the input signals with pull-up resistors LOW and input signals with pull-down resistors HIGH. While in the Stop Grant state, the pull-up resistors on STPCLK# and UP# are disabled internally. The system must continue to drive these inputs to the state they were in immediately before the CPU entered the Stop Grant state. For minimum CPU power consumption, all other input pins should be driven to their inactive level while the CPU is in the Stop Grant state.

2

3.2.2 GENERAL DESIGN CONSIDERATIONS

The STPCLK# input is an asynchronous signal. The system can crash if the interface to the STPCLK# is not designed properly. Special care must be taken to ensure that all the timing requirements are met and the proper protocol is used. Listed below are some design considerations that should be considered when designing the STPCLK# interface.

- The CPU cannot empty the write buffer during an HLDA cycle. Therefore, the CPU will not acknowledge any STPCLK# request during an HLDA cycle.

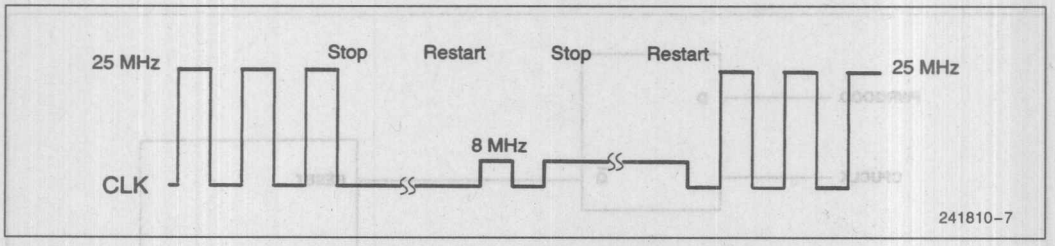


Figure 7. Clock Switching on IX Clock Input

- After the STPCLK# is asserted, the CPU does not generate a Stop Grant cycle until it completes the current instruction. The latency between a STPCLK# request and the Stop Grant bus cycle depends on the current instruction, the amount of data in the CPU write buffers, and the system memory performance.
- The CPU will not enter the Stop Grant state until either RDY# or BRDY# has been returned.
- In response to HOLD being driven active during the Stop Grant state (when the CLK input is running), the CPU will generate HLDA and three-state all output and input/output signals that are three-stated during the HOLD/HLDA state. After HOLD is de-asserted all signals will return to the state they were in prior to the HOLD/HLDA sequence.
- When the CPU enters the Stop Grant state, the internal pull-up resistor is disabled so that the CPU power consumption is reduced. The STPCLK# input must be driven high (not floated) in order to exit the Stop Grant state.
- It is the responsibility of the system designer to ensure that the CPU is in the correct state prior to asserting cache invalidation or interrupt signals to the CPU.

4.0 RESET IMPLEMENTATION

On a standard PC, the CPU can be reset by either hardware or software. On the SL Enhanced Intel486 CPU, asserting the RESET input to the CPU will also set the SMBASE register to the default value of 30000H. In other words, the SMRAM base address will reset to 30000H whenever the operating system asserts the CPU RESET signal. For some older software, a CPU RESET is generated by the software to return the CPU to real mode from protected mode.

Normally, this is not a problem if SMBASE relocation is not used. If the SMRAM base address has been relocated, the CPU could be executing invalid SMM code from the address. The SRESET pin has the same functions as RESET except that it does not reset the SMBASE register.

For a system which uses SMBASE relocation, the logic which generates the software CPU RESET must be tied to the SRESET input and not the RESET input on the CPU. All hardware resets should be implemented through the RESET pin (see Figure 8), and all software resets should be implemented through the SRESET pin.

While inside SMM, the CPU should be protected from being reset by a software CPU RESET. SRESET should be blocked whenever the SMIACT# is active. Any request for a CPU RESET when the CPU is in SMM should be latched so it can be serviced after the CPU exits SMM. To ensure the execution of the RSM instruction does not get interrupted by the SRESET, the SRESET must be blocked until at least 20 CPU clock cycles after SMIACT# has been driven inactive.

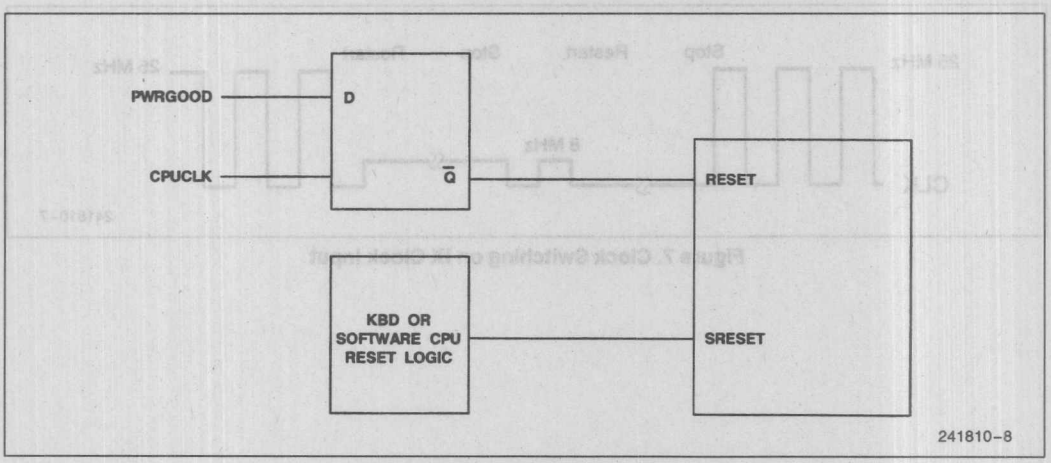


Figure 8. SRESET Interface Logic

4.1 General Design Considerations

The system designer should consider the following restrictions while implementing the CPU Reset logic:

- For SRESET to be recognized by the CPU, the SRESET input must be driven active (high) for at least 15 CPU clock cycles.
- The SRESET is not intended to be used for flushing the on-chip cache. For compatibility with future generation Intel CPUs, the SRESET input pin should not be used for flushing the on-chip cache.
- Hardware resets should not be blocked when the CPU is in SMM so that the system can recover from a fatal system failure.

CONCLUSION

While taking advantage of the benefits of the SL Architecture, the SL Enhanced Intel486 CPU family provides a whole new world of opportunity for system designers to develop innovative, energy-efficient mobile and desktop designs. The SL Enhanced Intel486 microprocessor family combines power management, compatibility and performance, allowing system designers to build a wide variety of machines to meet the diverse needs of a broad range of users.

References

Intel's SL Architecture: Designing Portable Applications, 1993, McGraw-Hill.

Intel486 Microprocessor Family Data Book Addendum: SL Enhanced Intel486 Microprocessor Family, 1993, Intel Corporation.

APPLICATION NOTE

Managing Power with the SL Enhanced Intel486™ Microprocessor

CHENG XIE
MCG TECHNICAL MARKETING

October 1993

Order Number: 241811-001

Managing Power with the SL Enhanced Intel486™ Microprocessor

CONTENTS

PAGE

1.0 INTRODUCTION 2-1056

2.0 POWER MANAGEMENT FEATURES 2-1056

2.1 System Management Mode 2-1056

2.2 Flexible Clock Control Options .. 2-1057

2.3 Different Low Power States 2-1057

2.3.1 Power States in 1X Mode .. 2-1057

2.3.2 Transition of Power States and Latency Associated with 1X Mode 2-1058

2.3.3 Power States in 2X Mode .. 2-1059

2.3.4 Transition of Power States and Latency Associated with 2X Mode 2-1059

CONTENTS

PAGE

3.0 IMPLEMENTATION OF POWER MANAGEMENT FEATURES 2-1060

3.1 CPU Power Control 2-1060

3.2 Controlling Power of Peripheral Components 2-1060

3.3 Suspend and Resume 2-1061

4.0 SUMMARY 2-1061

2

1.0 INTRODUCTION

Intel's System Management Mode (SMM), introduced as part of the Intel SL technology, has become an industry standard for portable computing. Through the utilization of SMM, system designers have a new method of adding software controlled features that operate transparently to the operating system and applications software. In portable computer systems, SMM is often used to implement the power control on various system components to conserve power consumption. Flexible clock control has also become essential to the design of power-saving computers. The new SL Enhanced Intel486™ microprocessor family incorporates all of the best power management features which first appeared in Intel SL technology, bringing Intel486 CPU performance to portable computers and energy-efficient desktop systems. The purpose of this application note is to provide system designers with a better understanding of the theory and implementation of power management with the new SL Enhanced Intel486 CPUs.

2.0 POWER MANAGEMENT FEATURES

- **System Management Mode**—This mode is composed of a special purpose interrupt that serves as the hardware interface and a secure memory address space that stores processor status and special software routines. It can be used to implement power management for portable systems.
- **Flexible Clocking Options**—The clock input to the CPU can either be a 1X clock or a 2X clock. For 1X clock option, the internal clock of the CPU runs at the same speed as the input clock (CLK input). In the 2X clock case, the clock input (CLK2 input) needs to be twice the frequency of the internal clock.
- **Different Low Power States**—Different low power processor states are available for various operating conditions. This feature enables the conservation of processor power consumption without sacrificing performance.
- **Low Voltage Power Supply Option**—The SL Enhanced Intel486 CPUs can be powered by either a 5V or a 3.3V supply, with the 3.3V supply providing a 50% power savings.

2.1 System Management Mode

The System Management Interrupt (SMI#) input pin on the processor provides the hardware interface for the computer system to invoke SMM. An exclusive memory address space, SMRAM, is only available for the CPU to access while in SMM. The size of SMRAM can be between 32 Kbytes and 4 Gbytes. It is used to store processor state and SMI handlers. SMI handlers are special software routines that can be designed to control the power states of system components. Intel's SMM has a special instruction, RSM, that is responsible for exiting SMM. When executed, RSM instruction restores the processor state from SMRAM and returns control to the application that was interrupted by SMI#.

The servicing of SMI# is different from that of a regular interrupt. The system invokes SMM by generating an SMI# to the SL Enhanced Intel486 CPU. Normal program execution will be interrupted as a result and the CPU will respond to the interrupt by asserting SMIACT#. This signal is used by the system to enable SMRAM space. The CPU will then save its state into the SMRAM area, starting at address location 3FFFFH and proceeding downward in a stack-like fashion. After completing a state save, the CPU will be in a pseudo real-mode processor environment. The microcode will then direct the CPU to begin executing instructions at the absolute address of 38000H (SMRAM), at which point it will begin executing SMI handlers. SMI handlers can perform various system management functions, including system power control. The last instruction in an SMI handler is always the RSM instruction. This instruction will restore the CPU state from SMRAM, de-assert SMIACT# and return control of the system to the interrupted program.

The generation of an SMI# to the CPU can be initiated by hardware or software for the purpose of power management. The actual implementation depends on the specific chipset used and how the system is designed. Hardware can generate SMI# by pulling the SMI# pin low directly or through other chipset pins, i.e., battery level control. When the charging level of the main battery falls below a certain limit, the chipset monitoring the battery level can interrupt the normal program operation by pulling the SMI# pin low. While in SMM, the CPU can execute certain power-down SMI handlers to put the entire computer system in a low-power mode that can operate out of a different battery source. This will enable the system to maintain its current status while allowing the main battery to be changed. SMI# can also be initiated through software. Different chipsets have various ways of interfacing to the CPU in this aspect. Most of them have dedicated timers to detect an idled device. Once these timers are enabled, the timeout will automatically generate an SMI#.

Exiting SMM is accomplished by the execution of the RSM instruction. Besides restoring the CPU state, the RSM instruction can also perform three other functions. The first is called "Auto HALT Restart". The System Management Interrupt request can interrupt the HALT instruction. By setting the appropriate control slot in the SMRAM space, the RSM instruction can return control to the HALT instruction or the in-

struction immediately following the HALT instruction. The second special feature is "I/O Trap Restart". If SMI# interrupt is generated on an I/O access, the RSM instruction will re-execute that I/O instruction if its I/O Trap Restart slot in the SMRAM is set. The third function is the relocation of SMBASE, the starting address of SMRAM. This provides system designers with the flexibility of placing SMRAM space into an area in which its integrity is best ensured. The starting address is controlled by the SMBASE register, which has a power-on reset value of 30000H. The default SMRAM area starts at 38000H and ends at 3FFFFH. The SMRAM can be relocated to any 32K aligned area, either overlaid on top of the normal system address space or placed in a distinct address space.

2.2 Flexible Clock Control Options

The standard Intel486 SX and Intel486 DX CPUs are driven by 1X clock as opposed to the Intel386 CPUs which use a 2X clock input. The SL Enhanced Intel486 CPUs are available with either the 1X or the 2X clocking options.

The 1X clock allows simpler system design by cutting in half the clock speed required in the external system. The 1X clock relies on an internal Phase Lock Loop to generate the two internal clock phases, "phase one" (ph1) and "phase two" (ph2). The rising edge of the CLK input corresponds to the start of ph1. All external timings are specified with respect to the rising edge of the CLK input. The PLL requires a constant frequency CLK input (to within 0.1%), and therefore the CLK input cannot be changed dynamically.

The 1X CLK input option is essential for those processors with an on-chip clock doubler. The 1X CLK input provides the fundamental timing references for the bus interface unit. The CLK input is doubled internally so that the CPU core will operate at twice the CLK input frequency, and hence twice the bus frequency. The internal clock doubler enhances all operations using the internal cache and/or not blocked by external bus accesses. This mode also uses PLL and therefore the CPU CLK input must be maintained at a constant frequency.

The SL Enhanced Intel486 CPUs also offer a 2X clock option for systems that rely on dynamic frequency scaling for CPU power management. The frequency of the CLK2 input is twice the internal frequency of the CPU. The internal clock is comprised of two phases, "PH1" and "PH2". Each CLK2 period is a phase of the internal clock. All timings are referenced to the rising edge of the PH1 of the CLK2 input. It is therefore important to synchronize the external circuitry with the PH1 of the CLK2 input. Because the 2X clock option does not rely on the PLL to generate the internal phase clocks, the frequency of the CLK2 input can be changed dynamically or "on-the-fly".

2.3 Different Low Power States

2.3.1 POWER STATES IN 1X MODE

Several low power modes are available on the 1X microprocessors. These modes make it possible for a power-sensitive computer system to optimize power conservation. Some of the CPU power controls are realized through a specially provided interrupt mechanism. Each of the following low power states is described in detail.

- **Auto Idle Power Down State**

  This low power state is available in normal operation for the DX2 CPUs. DX2 CPUs have an internal clock doubler which doubles the 1X clock input and therefore enables the CPU core to operate at twice the speed of the input clock. When the SL Enhanced Intel486 CPU is known to be truly idle and waiting for a ready from a memory or an I/O bus cycle read or write, the DX2 CPU will reduce its core clock rate to 1X from the doubled DX2 clock. In this state, the CPU only consumes half of the normal power. More importantly, this function is transparent to software and external hardware and therefore does not cause any performance degradation.

- **Stop Grant State**

  The Stop Grant state is initiated by simply asserting the external STPCLK# interrupt pin. The Stop Grant state is used to transition to the Stop Clock state.

  The CPU enters the Stop Grant state through the following steps: When the CPU recognizes the STPCLK# request, it will stop the execution of the normal program on the next instruction boundary, stop the pre-fetch unit, empty all internal pipelines and write buffers, generate a Stop Grant bus cycle and then stop the internal clock.

  This state is exited when STPCLK# pin is pulled high. The rising edge of the STPCLK# will tell the CPU to return control to the interrupted program and start to execute the instruction following the last executed instruction of the interrupted program.

- **Stop Clock State**

  The Stop Clock state is entered from the Stop Grant state by completely stopping the clock input to the PLL. In this state, the CPU consumes only ~100 μA–200 μA of current.

- **Auto HALT Power Down State**

  When the HALT instruction is executed, the CPU will issue a normal HALT bus cycle. The SL Enhanced Intel486 CPU will automatically stop the internal CPU clock, therefore causing the CPU to enter a low power state with a current of ~20 mA–55 mA.

- **Stop Clock Snoop State**

  Cache snooping is necessary during Stop Grant and Auto HALT Power Down states in order to maintain memory coherency. When the system issues a request for cache snooping, the CPU will transparently enter the Stop Clock Snoop state and will power up for 1 full core clock to complete the cache snoop cycle. It will then re-freeze the clock to the CPU core and either return to the Stop Grant or the Auto HALT Power Down state.

2.3.2 Transition of Power States and Latency Associated with 1X Mode

It is important to understand how the different power states are interrelated and how one transitions to another. The latency is different for different state transitions. Figure 1 depicts which transitions are allowed. We shall describe how the transitions are made and how much latency is associated with each transition.

1. The Auto Idle Power Down state is entered whenever the CPU is detected idle and waiting for a RDY# from either a memory or I/O read/write. This state only applies to SL Enhanced Intel486 DX2 CPUs. Both the internal CPU core clock and the current drop to half of the doubled frequency. There is no latency associated with this transition. The CPU will go back to normal operation when a RDY# is detected.
2. The Stop Grant state is entered when the STPCLK# interrupt is asserted to the CPU by the system. In this state, the clock output of PLL (or the clock input to the internal core) is stopped. The speed of the clock input to the PLL can be maintained or changed. If the clock frequency is changed, the CPU requires the clock to be held at a constant frequency for a minimum of 1 ms before de-asserting STPCLK#. The 1 ms time period is necessary so that the PLL can stabilize the input clock period. De-asserting STPCLK# returns the CPU to normal operation. The CPU will also return to its normal state when RESET or SRESET is asserted.
3. The Stop Clock state can only be entered from Stop Grant state when the clock input to the PLL is stopped. The CPU must go through Stop Grant state to return to normal operation. Before the CPU can return to Stop Grant state, the clock input has to stabilize for the 1 ms required by the PLL.
4. The Auto HALT Power Down state is entered when HALT instruction is executed. The clock input to the internal CPU core is automatically stopped upon the execution of HALT instruction. The clock input to the PLL cannot be changed during this state. This state is exited back to normal operation whenever any of the following events happen: INTR, NMI, SMI#, RESET or SRESET. There is no latency associated with this state transition.

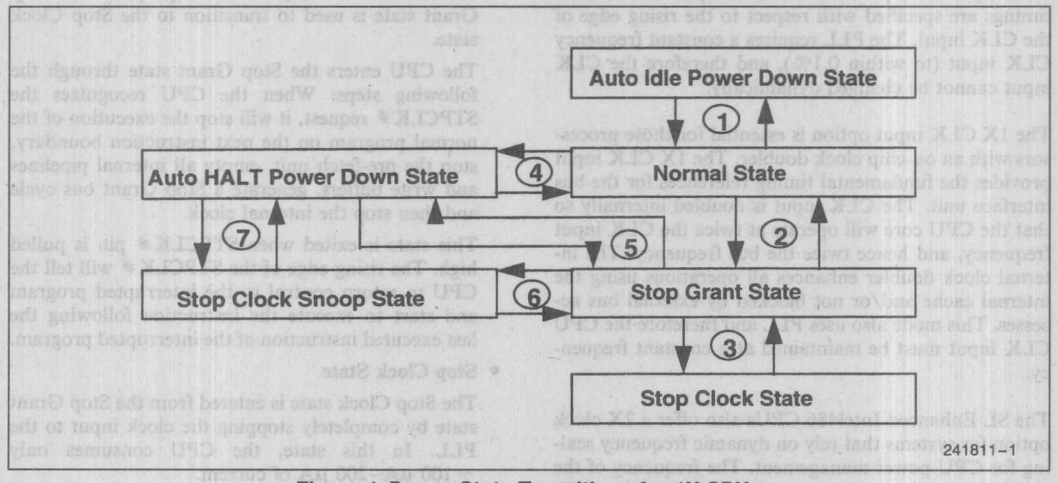


Figure 1. Power State Transitions for 1X CPUs

5. When the CPU is in Auto HALT Power Down state, the system can generate STPCLK# to the CPU to bring the CPU into a Stop Grant State. When the system de-asserts the STPCLK# request, the CPU will return to the Auto HALT Power Down state. There is no latency associated with this transition. HALT bus cycles will be launched whenever this transition occurs.

6, 7. Cache snooping can be performed when the CPU is either in Stop Grant state or in Auto HALT Power Down state. Cache snoop cycles begin when the CPU receives an EADS# from the system. The CPU will only wake up for 1 complete core clock to perform cache invalidation and then re-freeze the clock, i.e., either return to the Auto HALT Power Down state or to the Stop Grant state.

2.3.3 POWER STATES IN 2X MODE

There are five operating modes for the 2X SL Enhanced Intel486 CPU. The CPU power controls are realized through a specially provided interrupt mechanism, STPCLK#. In the following, we shall describe each of the power states in detail.

- **Normal State**

  2X CPUs do not have a PLL, and therefore do not require a stabilized clock period. This means the frequency of the input clock (CLK2) can be changed dynamically. Depending on the level of activity, CPUs do not always have to operate at full speed. Reducing the CPU speed saves power.

- **Stop Grant State**

  The Stop Grant state is initiated by simply asserting the external STPCLK# interrupt pin. The Stop Grant state is used to transition to the Stop Clock state.

  The CPU enters the Stop Grant state through the following steps: when the CPU recognizes the STPCLK# request, it will stop the execution of the normal program on the next instruction boundary, stop the pre-fetch unit, empty all internal pipelines and write buffers, generate a Stop Grant bus cycle and then stop the internal clock.

  This state is exited when STPCLK# pin is pulled high. The rising edge of the STPCLK# will tell the CPU to return control to the interrupted program and start to execute the instruction following the interrupted instruction.

- **Stop Clock State**

  The Stop Clock state is entered from the Stop Grant state by completely stopping the clock input (CLK2). In this state, the CPU consumes only ~100 μA–200 μA of current.

- **HALT State**

  When the HALT instruction is executed, the CPU will issue a normal HALT bus cycle. For 2X CPUs, there is no power savings in this state.

- **Stop Clock Snoop State**

  Cache snooping is necessary during Stop Grant state in order to maintain memory coherency. When the system issues a request for cache snooping, the CPU will transparently enter the Stop Clock Snoop state and will power up for 1 full core clock to complete the cache snoop cycle.

2.3.4 TRANSITION OF POWER STATES AND LATENCY ASSOCIATED WITH 2X MODE

Figure 2 shows the state transitions of a 2X microprocessor. We shall describe how the transitions are made and how much latency is associated with each transition.

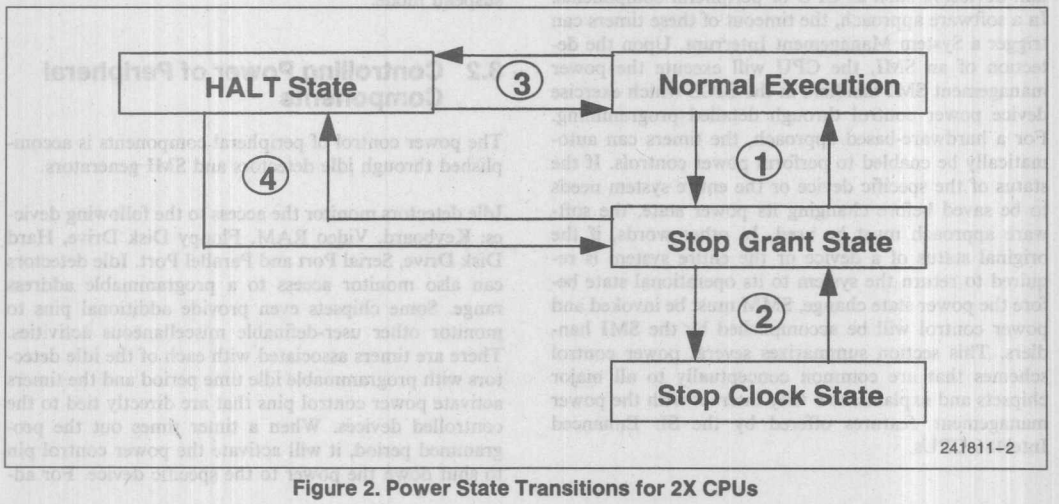


Figure 2. Power State Transitions for 2X CPUs

1. The Stop Grant state is entered when the STPCLK# interrupt is asserted to the CPU by the system. In this state, the speed of the external clock input (CLK2) can be maintained or changed. There is no latency associated with changing the CLK2 frequencies. De-asserting STPCLK# returns the CPU to normal operation. The CPU will also return to its normal state when RESET or SRESET is asserted.
2. The Stop Clock state is entered from Stop Grant state when the clock input (CLK2) is stopped. The CPU must go through Stop Grant state to return to normal operation. There is no additional delay associated with returning to normal operation.
3. This state is entered when the HALT instruction is executed. The HALT state consumes the same power as the normal state. The HALT state is exited back to normal operation whenever any of the following events happen: INTR, NMI, SMI#, RESET or SRESET. There is no latency associated with this state transition.
4. When the CPU is in HALT state, the system can generate STPCLK# to the CPU to bring the CPU into Stop Grant state. When the system de-asserts the STPCLK# request, the CPU will return to the HALT state. There is no latency associated with this transition. HALT bus cycles will be launched whenever this transition occurs.

3.0 IMPLEMENTATION OF POWER MANAGEMENT FEATURES

The means of implementing power management features depend on the specific chipset used. Most of the chipsets have both hardware and software power management. There are always a number of dedicated or user-definable timers that monitor the activity of certain device(s), such as CPU or peripheral components. In a software approach, the timeout of these timers can trigger a System Management Interrupt. Upon the detection of an SMI, the CPU will execute the power management SMI handlers in the BIOS which exercise device power control through detailed programming. For a hardware-based approach, the timers can automatically be enabled to perform power controls. If the status of the specific device or the entire system needs to be saved before changing its power state, the software approach must be used. In other words, if the original status of a device or the entire system is required to return the system to its operational state before the power state change, SMM must be invoked and power control will be accomplished by the SMI handlers. This section summarizes several power control schemes that are common conceptually to all major chipsets and explains how they interact with the power management features offered by the SL Enhanced Intel486 CPUs.

3.1 CPU Power Control

Controlling SL Enhanced Intel486 CPU power is achieved by reducing the CPU clock speed, stopping the CPU clock or shutting off the CPU power.

All chipsets have the option of pre-programming the CPU speed regardless of the level of system activity there is and the CPU clock speed can be divided down by 2 to 64. Once the CPU is selected to run in a reduced speed mode and the CPU clock division is selected, the CPU will always run at a divided speed until the CPU is switched into some other mode of operation. Speed reduction is done by BIOS through programming certain register bits immediately after booting.

The speed of the CPU can also be changed dynamically depending on the level of system activity. Most of the chipsets provide mechanisms of monitoring the level of system activity involving the CPU by detecting the toggling of certain CPU signal lines. The timers associated with these monitoring devices are responsible for determining when to reduce the CPU speed by timing out a programmable time period. The chipset reduces the speed of the CPU clock by dividing its clock input to the CPU after STPCLK# is asserted. For 2X SL Enhanced Intel486 CPUs, the speed can be changed on-the-fly. For 1X SL Enhanced Intel486 CPUs, clock input has to be stabilized for 1 ms before de-asserting STPCLK#. Stopping the CPU clock is accomplished in the same fashion.

The CPU clock control can be achieved either through hardware or software approach. If the status of the CPU needs to be saved in order to return the system to the original state, CPU clock control should be done through the SMI handlers in SMM mode.

Shutting off the power to CPU can only be done in suspend mode.

3.2 Controlling Power of Peripheral Components

The power control of peripheral components is accomplished through idle detectors and SMI generators.

Idle detectors monitor the access to the following devices: Keyboard, Video RAM, Floppy Disk Drive, Hard Disk Drive, Serial Port and Parallel Port. Idle detectors can also monitor access to a programmable address range. Some chipsets even provide additional pins to monitor other user-definable miscellaneous activities. There are timers associated with each of the idle detectors with programmable idle time period and the timers activate power control pins that are directly tied to the controlled devices. When a timer times out the programmed period, it will activate the power control pin to shut down the power to the specific device. For ad-

dress range idle detectors, when there is no access to the monitored address range for a programmed timeout period, the timers will either reduce the clock speed or shut off the power of the device that has the same address bits. The idle detectors can operate outside of SMM mode and can be independent of CPU state.

SMI event generators generate SMI requests through a number of dedicated or user-defined events. Depending on the specific chipset used, these events can be the activity timeout of individual devices or a collection of devices. Upon the timeout, an SMI request will be generated to the CPU, which in turn invokes SMM. The SMI handlers in the SMRAM contain the software routine that controls the power state of the device(s) initiating the SMI request. By executing this routine, the CPU will access the power management control registers associated with the devices. After proper programming, these registers will activate the proper power control pins to shut down the power to the proper device(s). Controlling the power of peripheral devices through SMI handlers offers complete flexibility to either manage the power individually or collectively. This is very important for optimum power conservation.

3.3 Suspend and Resume

Suspend state is the deepest level of power conservation. There are two types of suspend: normal suspend and 0-volt suspend. In normal suspend, only the CPU, chipset and memory sub-system are powered. System status is saved into SMRAM. The rest of the system is shut down. DRAM is refreshed with a very slow clock (64 KHz or 32 KHz). In 0V suspend, the entire system, including the CPU, is shut down except the part of the system logic that is responsible for resume. The resume logic is always powered by an RTC battery. The system status is saved onto hard disk. Suspend is normally triggered by the suspend timer, and the timeout of the suspend timer is also programmable.

Because of the amount of BIOS support required by Suspend, SMM must be invoked. Hardware alone cannot accomplish the task.

4.0 SUMMARY

SL Enhanced Intel486 CPUs provided the best power management features that Intel SL technology offers. Intel's System Management Mode has become an industry standard for power-saving computing. Through Intel's SMM, the implementation of power management is very flexible, enabling the optimization of power conservation for different system designs. The various CPU clock control options available on SL Enhanced Intel486 processors provide the basis for runtime power management with no performance impact. All major chipsets support the Intel power management scheme with easy-to-design software and hardware interfaces.

intel®

AP-498

APPLICATION NOTE

Thermal Design for High Performance Notebooks

VLADIMIR ALEKSIEV
CHIA-PIN CHIU
WENDY LUI
ED WILSON
DAVID YUAN
MCG TECHNICAL MARKETING
MCG PACKAGING
THERMAL/MECHANICAL TOOLS AND ANALYSIS GROUP

November 1993

Order Number: 241812-001

Thermal Design for High Performance Notebooks

CONTENTS

PAGE

1.0 INTRODUCTION 2-1064

2.0 THERMAL BACKGROUND 2-1064

2.1 Heat Transfer 2-1064

2.1.1 Conduction 2-1064

2.1.2 Convection 2-1064

2.1.3 Radiation 2-1065

2.2 Thermal Impedance 2-1065

3.0 POWER MODELING 2-1066

3.1 Power Consumption Model 2-1066

3.2 Empirical Data 2-1066

3.3 Typical System Power Consumption Profiles 2-1067

4.0 θ_{JA} BASED ON EXTERNAL T_A ... 2-1067

4.1 Measurements from Commercial Notebooks 2-1067

4.2 θ_{JA} and θ_{JC} Measurements in an Actual Test Environment 2-1068

4.3 System Impact on CPU θ_{JA} 2-1069

CONTENTS

PAGE

5.0 CALCULATING THERMAL HEADROOM 2-1071

5.1 Using Thermal Headroom Graphs 2-1071

5.2 Example of Graph Use 2-1071

5.3 Adjusting Thermal Headroom for Board Power 2-1071

5.4 Experimental Measurements are Essential 2-1072

5.5 Secondary Effects 2-1072

6.0 IMPROVING THERMAL HEADROOM 2-1074

6.1 Improving Thermal Convection, Conduction and Radiation 2-1074

6.1.1 Black Paint 2-1074

6.1.2 Copper Foil 2-1074

6.1.3 Perfluorocarbon Fluid and Silicone Elastomers 2-1074

6.2 Optimizing System Layout 2-1075

6.3 Effective System Power Management 2-1075

7.0 CONCLUSION 2-1076

APPENDIX A 2-1077

APPENDIX B 2-1080

APPENDIX C 2-1081

1.0 INTRODUCTION

Today's market for notebook computers demands desktop performance in smaller and smaller form factors. Along with the higher performance comes greater power consumption, which adds unique challenges for the mobile operating environment (battery operating time, thermal management, physical dimensions, etc.). Included in this application note is a basic description of the thermal forces at work in mobile applications, with mathematical models that can be used to project system thermal parameters and aid the designer in worst-case design.

The first section begins with a review of the basic thermal definitions which apply to notebook designs. Thermal data from a notebook experiment is presented to show a relationship between the temperature outside the notebook and the CPU case temperature. A model is given that helps the designer ensure the CPU thermal operating specifications are met.

Next, several power consumption profiles are provided, starting with the assumed worst-case model, along with more conservative power profiles based on the degree of power management implementation. Based on these models, designers may forecast the amount of additional thermal margin their applications need.

With these thermal models, power consumption profiles, and test measurements, the designers will have the necessary tools and techniques to design for the worst case, and understand that by applying simple design enhancements, they can improve the quality of their designs. The designer should ensure the measured CPU case temperature (T_{CASE}) complies with the T_{CASE} specifications published in the SL Enhanced Intel486 Microprocessor Data Sheet Addendum.

2.0 THERMAL BACKGROUND

2.1 Heat Transfer

Designing high performance CPU notebook systems requires some knowledge of the three processes by which heat is transferred from one point to another, namely: conduction, convection, and radiation, which are described in the following three sections. This knowledge will help the designer understand the subsequent methods of heat transfer and their value in maintaining the CPU within its specified T_{CASE} limits. The formulas describing heat transfer by conduction, convection and radiation can be shown analogous to Ohm's Law:

$$I = \frac{V}{R}$$

The thermal current, temperature difference and thermal resistance are analogous to electrical current (I), voltage (V), and electrical resistance (R), respectively.

2.1.1 CONDUCTION

Conduction is a process by which heat flows from a region of higher temperature to one of lower temperature within a medium (solid, liquid, or gas) or between mediums in direct physical contact[1]. In a one-dimensional system, conductive heat transfer is governed by the following relation:

$$\underbrace{q = \frac{\Delta T}{L/kA}}_{\text{Conductive Heat Transfer}} \rightarrow \underbrace{I = \frac{V}{R}}_{\text{Ohm's Law}}$$

where:

q = Heat flow rate (W)

k = Material thermal conductivity (W/m°C)

A = Cross-sectional area (m^2)

$\frac{\Delta T}{L}$ = Temperature gradient (°C/m)

L = Distance of heat transfer

In the preceding equations, the thermal current, q, can be viewed analogous to electrical current; ΔT analogous to voltage; and L/kA analogous to thermal resistance. To improve thermal conduction in a notebook environment, copper and other highly-thermal conductive metals can be used in the package design.

2.1.2 CONVECTION

Convection is a process of energy transport by the combined action of heat conduction, energy storage, and mixing motion[2]. Convection is the predominant mechanism for transferring energy between a solid surface and a fluid. In the notebook environment, this is equivalent to the heat transfer between the case surface and the ambient environment (air). The basic relation that describes heat transfer by convection from a surface to a fluid presumes a linear dependence on the difference between the temperature at the surface and deep in the fluid, and is referred to as Newtonian cooling:

$$\underbrace{q_C = \frac{T_S - T_A}{1/h_C A}}_{\text{Convective Heat Transfer}} \rightarrow \underbrace{I = \frac{V}{R}}_{\text{Ohm's Law}}$$

NOTES:

1. *Physics, Second Edition,* Paul A. Tiper. Worth Publishers, Inc., 1982, p. 531.

2. *Physics, Second Edition,* Paul A. Tiper. Worth Publishers, Inc., 1982, p. 531.

where:

q_C = Convective heat flow rate from a surface to ambient (W)

A = Surface area (m^2)

T_S = Surface temperature (°C)

T_A = Ambient temperature (°C)

h_C = Average convective heat transfer coefficient ($W/m^2°C$)

In the preceding equations, the thermal current, q_C, can be viewed analogous to electrical current; $T_S - T_A$ analogous to voltage; and $1/h_C A$ analogous to thermal resistance.

In forced convection, fluid flow is caused by an external factor such as a fan, while in free or natural convection, fluid motion is induced by density differences resulting from temperature gradients in the fluid (liquid or gas). Under the influence of gravity or other body forces, these density differences give rise to buoyancy forces that circulate the affected fluid and convect heat toward or away from surfaces wetted by the fluid. Although fans are often used to increase air convection inside desktop computers, they may not be a practical solution for notebook systems.

2.1.3 RADIATION

Thermal radiation is defined as radiant energy emitted by a medium by virtue of its temperature, without the aid of any intervening medium[3]. The amount of heat transferred by radiation between two bodies at temperatures T_1 and T_2 is governed by the following expression:

$$\underbrace{q = \frac{T_1^4 - T_2^4}{1/\epsilon\sigma}}_{\text{Radiative Heat Transfer}} \rightarrow \underbrace{I = \frac{V}{R}}_{\text{Ohm's Law}}$$

where:

q = Amount of heat transferred by radiation (W)

ϵ = Emissivity $0 < \epsilon < 1$

σ = Stefan-Boltzmann constant, 5.67×10^{-8} ($W/m^2 °K^4$)

NOTE:

3. *Physics, Second Edition,* Paul A. Tiper. Worth Publishers, Inc., 1982, p. 535.

A = Area (m^2)

T_1, T_2 = Surface temperature (°K)

For radiation to be an effective method of heat transfer, compared to natural or forced convection mechanisms, a relatively large temperature difference must exist between T_1 and T_2. For most low-power electronic applications, these temperature differences are relatively small, and therefore, radiative effects are normally neglected. However, for high-power applications, heat transfer by radiation factors should be considered. Although heat radiation is a secondary thermal effect in a notebook system, some manufacturers are selecting coatings (i.e., black paint) for their notebook designs that are absorptive in the infrared to improve upon these thermal radiation effects.

2

2.2 Thermal Impedance

Thermal management of an electronic system encompasses all the thermal processes and technologies which must be used to remove and transport heat from individual components to the system thermal sink in a controlled manner. The primary heat transfer processes (conduction, convection and radiation) can be combined into a single linear model (see Section 5.1).

The junction-to-case (θ_{JC}) and junction-to-ambient (θ_{JA}) thermal resistance values are used as measures of IC package thermal performance. These parameters are defined by the following relations:

$$\theta_{JC} = \frac{T_J - T_C}{P}$$

$$\theta_{JA} = \frac{T_J - T_A}{P}$$

$$\theta_{JA} = \theta_{JC} + \theta_{CA}$$

where:

θ_{JA} = Junction-to-ambient thermal resistance (°C/W)

θ_{JC} = Junction-to-case thermal resistance (°C/W)

θ_{CA} = Case-to-ambient thermal resistance (°C/W)

T_J = Average die temperature (°C)

T_C = Case temperature at a predefined location (°C)

P = Device power dissipation (W)

T_A = Ambient temperature (°C)

θ_{JC} is a measure of package internal thermal resistance from silicon die to package exterior. This value is highly dependent upon packaging material, thermal conductivity, and package geometry. θ_{JA} measures the

conductivity and convective thermal resistance from package exterior to the ambient, as well as package internal thermal resistance. θ_{JA} values depend on material, thermal conductivity, package geometry, and ambient conditions such as flow rates and coolant physical properties.

To improve CPU thermal characteristics in a notebook system, heat sinks are sometimes mounted on the top of the CPU using highly conductive adhesive materials. Adding a heat sink will not change θ_{JC}; however, it will improve heat conduction and convection due to the increase in surface area, resulting in a significant reduction in the case-to-ambient and, therefore, junction-to-ambient thermal resistance. Depending on the product and materials used, θ_{JA} can be reduced by 10 to 30 percent.

To guarantee component functionality and reliability, the maximum device operating temperature is defined and constrained by the package exterior temperature at a predefined location. The guidelines for ambient temperature specify that measurements should be taken at an undisturbed location at a certain distance away from the package—traditionally 12 inches horizontally from the center of the CPU. Measuring T_A in the traditional manner is not possible in a notebook system. The case temperature, however, is measured at the center surface of the package. Depending on the ambient temperature and board power in the system's environment, thermal enhancements such as heat fins or forced air cooling may be necessary to meet the case temperature requirements.

3.0 POWER MODELING

3.1 Power Consumption Model

In a linear model, power consumption is governed by the following equation:

$$P = V_{CC} \times I_{CC}$$

where:

P = Power consumed by the component
V_{CC} = Supply voltage
I_{CC} = Current through the component

The preceding equation indicates that power consumption is linearly proportional with both supply voltage and current flowing through the component. For example, a 3.3V microprocessor will consume less power than a 5V microprocessor running the same application under equivalent operating conditions.

A system's total power consumption is defined as either the sum of the power consumed by each individual module within the system, or the sum of the products of each module's voltage supply and its current. It is equivalent to:

$$P_{system} = P_{CPU} + P_{memory} + P_{display} + \text{etc}$$

3.2 Empirical Data

Several CPU I_{CC} measurements were taken using an SL Enhanced Intel486 CPU evaluation board with various CPU modules running under two different operating environments: Windows 3.1 and Indeo™ Video software (see Table 3-1). The modules used were SL Enhanced Intel486 DX-33 CPUs (1X SQFP, 1X PGA, and 2X SQFP), and SL Enhanced Intel486 DX2-50 CPUs (1X PGA). All of the CPU modules use a chipset with Power Management software.

When comparing the I_{CC} measurements between several application environments, the CPUs running Indeo Video software consume the most I_{CC} current by approximately 10%. Since the I_{CC} value represents the number of gates switching inside the CPU, and hence, the intensity of the CPU working condition, it is therefore concluded that running Indeo Video software will be close to the worst case for thermal measurement purposes.

Table 3-1. Case Analysis of Power Consumption for SL Enhanced Intel486 CPUs

							Windows 3.1			Indeo™ Video Software		
							I_{CC}			I_{CC}		
CLK	CPU	V_{CC}	Freq	Package	θ_{JC}	θ_{JA}	Active	Stop Grant	STPCLK	Active	Stop Grant	STPCLK
1X	DX	3.3	33	SQFP	3.5	25.0	0.279	0.008	0	0.290	0.008	0
	DX	5.0	33	PGA	1.5	17.0	0.491	0.041	0	0.512	0.041	0
	DX2	5.0	50	PGA	1.5	17.0	0.656	0.046	0	0.685	0.046	0
2X	DX	3.3	33	SQFP	3.5	25.0	0.283	N/A	0	0.294	N/A	0

NOTES:
1. All thermal values are measured at zero airflow.
2. Measurements taken on an SL Enhanced Intel486 CPU Evaluation Board (no heat spreader, no heat sink).
3. All measurements were taken using one module of each microprocessor.

3.3 Typical System Power Consumption Profiles

Table A-2 in the Appendix examines the power dissipated for four typical "system" profiles. These are systems in the sense that the power used is assumed controlled (except in the first case) by either the operating system or the system hardware, aside from the CPU. These cases, however, do **not** attempt to add the effects of other power dissipating components that would exist in a complete PC notebook system. The first case gives the most conservative power calculation: the maximum power that can be generated by the CPU. The second case gives the typical average power. The last two suggest power consumption possibilities that could occur in a given system, or even be guaranteed by power management. Many similar combinations would also be reasonable. V_{CC} in each case assumed as the standard value (5.0V or 3.3V).

In Case 1, the average power is calculated as the standard V_{CC} (3.3V or 5V) times the maximum current, I_{CC}(max), that can be drawn by a particular CPU. This calculation gives the maximum power that can be dissipated while continuously executing the most power consuming instruction sequence. This value should be used in a system design if no thermal power management is imposed, and the designer wants to minimize potential problems even under worst-case circumstances: a conservative design.

In Case 2, the average power is calculated as the standard V_{CC} (3.3V or 5V) times the TYPICAL current, as specified in Intel486 microprocessor data books. This calculates the average heat from the CPU that would be dissipated over time while executing a typical mix of software. The designer could use this power value in a less conservative design. However, if the CPU case temperature approaches its maximum specified value and no thermal power management is applied, the T_C(max) specification could be exceeded. (Section 5.5 discusses the time dependency issues in averaging the thermal power generated while executing a mix of software.)

In Case 3, the average power is calculated as the standard V_{CC} (3.3V or 5V) times the I_{CC}(max) for 10% of the time, and I_{CC}(typical) for 90% of the time. This is a more conservative assumption than Case 2, allowing for some intervals in which the CPU runs at full power, and an overall thermal guardband over Case 2.

Case 4 assumes that the current is distributed at the maximum for 10%, typical for 80%, and Stop Clock for 10% of the time. A mix of this sort is appropriate in a design where power management is applied to assert Stop Clock for at least 10% of the time. This mix could reduce the performance of the CPU. Suppose the system designer devises a theoretical mix of I_{CC}(Max, Typical and Stop Clock) which is exceeded by the system only 1% of the time when running all standard benchmarks and applications. The designer builds the system to that specification, with thermal power management responding only when the limit is exceeded. Then one can safely design the system assuming a significantly lower power than the absolute maximum, and experience performance degradation only 1% of the time.

The cases above illustrate that many less than worst-case power profiles are possible, depending on the software being run, and power management options being used. Intel recommends that systems be tested for thermal problems under the worst case power usage that the customer could contrive, and in an ambient temperature equal to the maximum specified for the design.

4.0 θ_{JA} BASED ON EXTERNAL T_A

2

This section provides some experimental thermal data which will help the designer better understand some of the system level issues affecting the thermal performance of a notebook. Section 4.1 describes in one experiment how test chamber θ_{JA} can be used as an approximate thermal performance criteria in the early stages of a notebook design. Section 4.2 shows how in another experiment θ_{JA} is affected by CPU location on the motherboard inside the notebook. Section 4.3 presents a model showing the relationship between power generated by other components on the motherboard and the CPU's θ_{JA} requirement.

4.1 Measurements from Commercial Notebooks

Since component location differs among notebook designs causing internal power densities to vary between notebooks there is no one place inside a notebook where T_A can be defined in order to obtain an accurate system thermal profile. This section presents experimental thermal data collected for four different Intel386 SL CPU notebooks running Indeo Video software and shows—as a rough estimate—that the ambient temperature (T_A) outside a notebook can be used with thermal resistance (θ_{JA}) to project CPU temperture (T_J, T_C).

In this notebook experiment, the CPU case temperature of four different Intel386 SL CPU notebooks (PQFP packages) were measured using K-type thermocouples a digital multimeter, and an Intel386 DX CPU-based system with data acquisition software. To simulate maximum I_{CC} consumption, each notebook continuously executed Indeo Video software for the duration of the experiment. Table 4-1 shows the junction temperature and θ_{JA} calculated by using the thermal impedance equations from Section 2.2. The CPU case temperature, the maximum power consumption of 2.5W

running Indeo Video software, and the test chamber θ_{JC} of 6°C/W as specified in the Intel386™ SL Microprocessor Data Book for the 196L PQFP were used to calculate the junction temperature of each CPU. With this calculated CPU T_J and measured ambient room temperature of 25°C, the corresponding notebook's θ_{JA} was obtained. θ_{JA} can be calculated by combining the first two equations in Section 2.2 as follows:

$$\theta_{JA} = \theta_{JC} + \frac{T_C - T_A}{P}$$

Although the test chamber θ_{JA} for the 196L PQFP of 23°C/W was collected with only the CPU present on a test board, the value obtained approximates θ_{JA} of a notebook operating in an environment of 25°C. One possible explanation is that the conductive and radiative effects the other components have on the CPU inside the operating notebook, such as the PC board, connectors, floppy disk, shielding and plastic enclosure, actually cause the temperature inside the notebook to be lower than expected. The end result is a lower θ_{JA} for the CPU inside the operating notebook than the θ_{JA} of a lone CPU inside a test chamber where the only conductive and radiative path is to the surrounding air, as shown in Notebooks #1 and #3. Table 4-1 shows the large variation in notebook θ_{JA} caused by different system designs. Thus, the test chamber θ_{JA} can be used as a rough guideline in the early stages of a notebook design. For a more in-depth analysis of the conditions inside an operating notebook, see Section 4.3. For the final design, the thermal performance of the system should always be verified by measuring the CPU case temperature.

4.2 θ_{JA} and θ_{JC} Measurements in an Actual Environment

A test motherboard with the same form factor as that of the original Test Notebook #4 was fabricated in order to take experimental measurements of θ_{JA} and θ_{JC} (see Figure 4-1). The only differences between the two boards are the following:

1. The test board had two slots instead of the 70/80 pin connectors on the motherboard.
2. The test board only had six thermal test packages mounted on it (three on the component side and three on the solder side).

Measurements from the 6 thermal test units yielded a θ_{JA} range of 20°C/W–25°C/W in the test chamber and 22°C/W–28°C/W in the Test Notebook and a θ_{JC} range of 3°C/W–5°C/W in both the test chamber and the Test Notebook (see Table 4-2).

Table 4-1. θ_{JA} Calculations for Intel386 SL CPU Notebooks Running Indeo™ Video Software

Notebook #	T_{CASE}	T_J (calculated)	θ_{JA} (calculated)
1	48.9	63.9	15.5
2	69.0	84.0	23.6
3	52.1	67.1	16.8
4	71.5	86.5	24.6

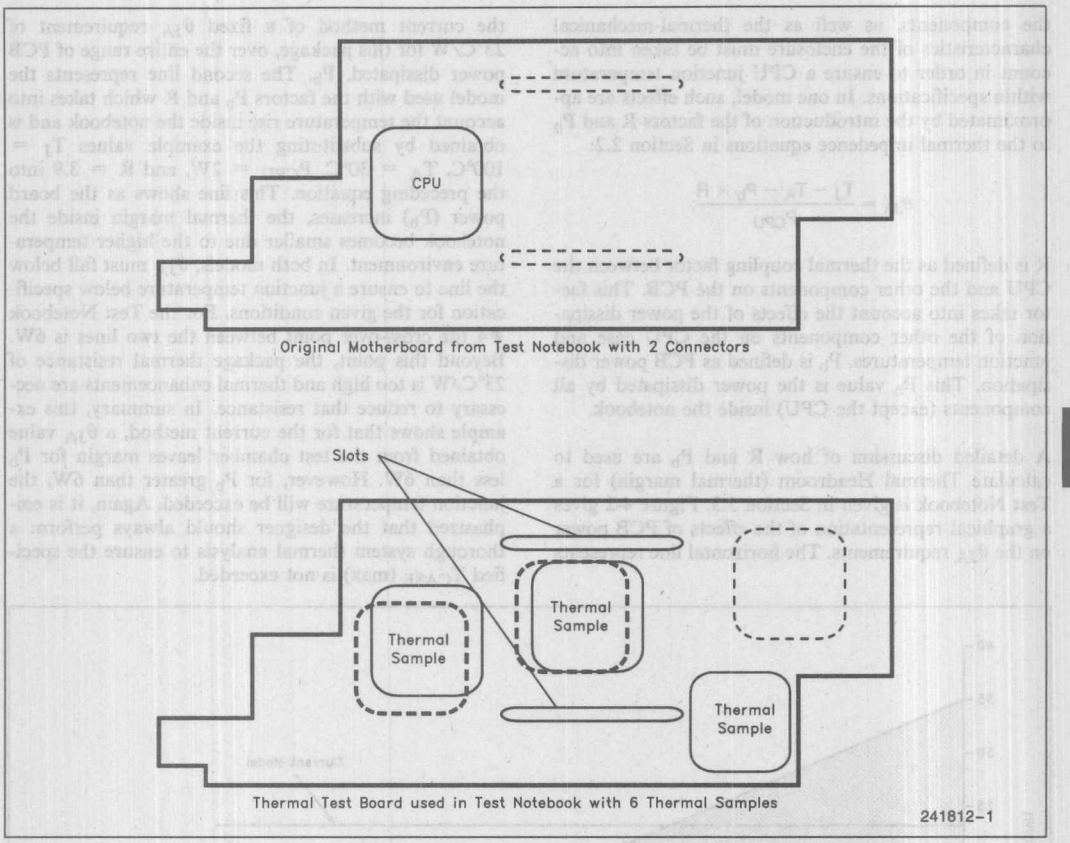


Figure 4-1. Test Notebook Boards

Table 4-2. Thermal Resistance θ_{JA} and θ_{JC} for the SQFP Package

	Test Motherboard in Test Chamber (°C/W)	Test Motherboard in Test Notebook (°C/W)
θ_{JA}	20–25	22–28
θ_{JC}	3–5	3–5

NOTES:
1. Test Notebook #4 was used to collect the experimental data.
2. 208L SQFP test package with heat spreader containing thermal test die was used in all experiments to vary and measure the power going into package as well as to measure the temperature of the die.
3. All measurements were made with zero airflow, simulating a typical notebook environment. Ambient temperature is defined as ambient temperature outside the notebook.

The worst CPU thermal location was on the center of the solder side with θ_{JA} = 28°C/W. The measured thermal resistance at this location was unfavorable because of the reduced CPU board area surrounding the CPU (due to the two slots) and the reduced convection cooling on the bottom side of the board. The best CPU thermal location was on the component side at one end of the test board with the most PCB area surrounding the package with θ_{JA} = 22°C/W. This shows how CPU location and system layout can impact the overall thermal performance of a notebook.

The θ_{JA} numbers should only be used as a first order estimate in preliminary notebook designs. Since the location of a CPU inside a notebook impacts thermal performance, T_J should always be verified in the final design by θ_{JC} and the CPU case temperature.

4.3 System Impact on CPU θ_{JA}

As notebooks evolve into smaller form factors with higher component density and smaller PCB sizes, the increasing power density inside the notebook has a large effect on CPU temperature. The power dissipated by components other than the CPU, and the layout of

the components, as well as the thermal-mechanical characteristics of the enclosure must be taken into account in order to ensure a CPU junction temperature within specifications. In one model, such effects are approximated by the introduction of the factors R and P_b to the thermal impedence equations in Section 2.2:

$$\theta_{JA} = \frac{T_J - T_A - P_b \times R}{P_{CPU}}.$$

R is defined as the thermal coupling factor between the CPU and the other components on the PCB. This factor takes into account the effects of the power dissipation of the other components on the CPU case and junction temperatures. P_b is defined as PCB power dissipation. This P_b value is the power dissipated by all components (except the CPU) inside the notebook.

A detailed discussion of how R and P_b are used to calculate Thermal Headroom (thermal margin) for a Test Notebook is given in Section 5.3. Figure 4-2 gives a graphical representation of the effects of PCB power on the θ_{JA} requirements. The horizontal line represents the current method of a fixed θ_{JA} requirement of 23°C/W for this package, over the entire range of PCB power dissipated, P_b. The second line represents the model used with the factors P_b and R which takes into account the temperature rise inside the notebook and is obtained by substituting the example values T_J = 100°C, T_A = 30°C, P_{CPU} = 2W, and R = 3.9 into the preceding equation. This line shows as the board power (P_b) increases, the thermal margin inside the notebook becomes smaller due to the higher temperature environment. In both models, θ_{JA} must fall below the line to ensure a junction temperature below specification for the given conditions. For the Test Notebook #4 the cross-over point between the two lines is 6W. Beyond this point, the package thermal resistance of 23°C/W is too high and thermal enhancements are necessary to reduce that resistance. In summary, this example shows that for the current method, a θ_{JA} value obtained from the test chamber leaves margin for P_b less than 6W. However, for P_b greater than 6W, the junction temperature will be exceeded. Again, it is emphasized that the designer should always perform a thorough system thermal analysis to ensure the specified T_{CASE} (max) is not exceeded.

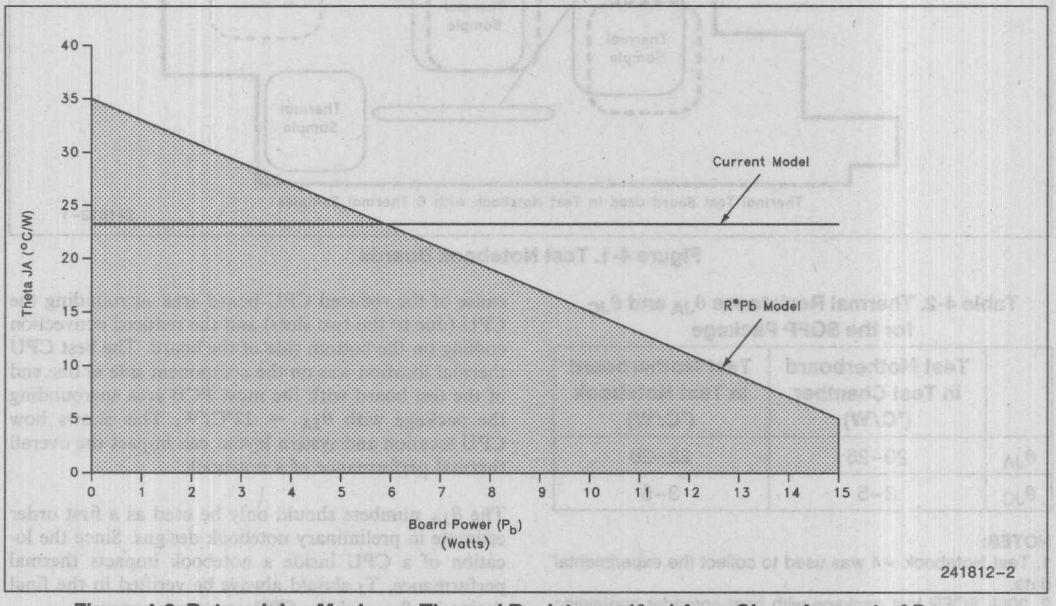


Figure 4-2. Determining Maximum Thermal Resistance (θ_{JA}) for a Given Amount of Power

5.0 CALCULATING THERMAL HEADROOM

Thermal Headroom is the temperature margin between the calculated T_A(max) and the T_A measured outside a given system, with a particular CPU and power. This section shows how to calculate Thermal Headroom and use it as simple model for a system's thermal properties. Then a term accounting for board power is added to the model, and the experimental measurements needed to implement this more sophisticated version are described. Finally, the significance of two secondary effects is analyzed.

5.1 Using Thermal Headroom Graphs

Figures A-1 and A-2 (in Appendix A) plot the calculated T_A(max) vs the power being used by the CPU and are intended to facilitate quick determination of thermal headroom. The lines on the graphs indicate the (estimated) maximum allowed ambient temperature (T_A in degrees Celsius) as a function of power dissipated (P in Watts). Maintaining T_A below or equal to T_A(max) indicates that the required T_C(max) is probably not exceeded. The graph lines are generated from the formula:

$$T_A(max) = T_C(max) - \theta_{CA} \times P$$

θ_{CA} is the thermal resistance to heat flow between the CPU case and the ambient environment, as specified in the Intel Packaging Handbook. As discussed in Section 4, experiments show that these parameters are approximately the same for a notebook PC with T_A measured in open air outside the notebook case. The tendency of the notebook case to *increase* θ_{CA} by adding extra layers of insulation is approximately offset by its action as a heat spreader, since it is thermally connected to the CPU board. Different types of CPU packages have different θ_{CA} values, and thus generate different lines on the graphs. For example, Figure A-1 shows the SQFP, PGA and PQFP packages for the SL Enhanced Intel486 SX CPU and Figure A-2 shows the SQFP and PGA packages for the SL Enhanced Intel486 DX2 CPU.

To determine thermal headroom for a given CPU, first determine the correct line for the CPU type. (Some of the CPUs are marked on the graphs. For a CPU type that is not, find its θ_{CA} in Table A-1, and match it to a CPU type that is marked on a graph line. Or use θ_{CA} as the slope, and T_C = 85°C as the temperature axis intercept to match directly to a graph line.)

Second, determine the power at which the CPU will operate. Various average power use scenarios could be appropriate for a given design; four of them are detailed in Table A-2 for each different CPU (discussed in detail in Section 3.3). One can also calculate a custom power usage profile for one's system using $P = I_{CC} \times V_{CC}$, and Table A-1, which gives I_{CC} under 3 different conditions (Active, Stop Grant and Stop Clock).

Third, draw an ordinate (vertical line) at the power value (determined above) to intersect the appropriate line for the CPU package type chosen. Draw an abscissa (horizontal line) from the intercept to the T_A axis, obtaining the maximum recommended ambient temperature for this system. If this $T_{A(max)}$ is greater than what is measured in the air outside the actual system, the design has a positive thermal headroom of $T_{A(max)} - T_A$(measured), as long as the effect of "board power", P_b, is neglected. If however the actual system is exceeding this T_A(max), the thermal headroom is negative even before considering P_b, and thus the thermal properties of the design will need improvement ("thermal mitigation"), (See Section 4 for P_b definition, and Section 5.3 for more information about Thermal Headroom).

There are numerous techniques for thermal mitigation, or improving the thermal properties of a design (i.e., a lower voltage version of the CPU or a CPU package with a lower θ_{CA} could be used). Depending on package size and material, various θ_{CA} values can be obtained. Various passive and active thermal management strategies are discussed in Section 6.

5.2 Example of Graph Use

Consider the 5V PQFP SL Enhanced Intel486 SX-33 CPU, which is likely to have special thermal needs because of its power requirements. The graph line for it is indicated by the label (33 MHz PQFP 5V) on the Intel486 SX CPU graph, or by the fact that its slope value from Table A-1 is 17.0 °C/W. The power shown for Figure A-1 is for Case 1: V_{CC} = 5.0V; I_{CC} = 0.685 mA (from Table A-1); Active Max for 100% of the time; P = 3.43W (from Table A-2, or calculation).

The ordinate is drawn from the P axis at 3.43W to intersect the intermediate slope line, and the abscissa from that point intersects the Temp. axis at about 27°C. If we assume 40°C is the lowest T_A that can be readily achieved, we get a negative thermal headroom of 13°C. Designing a portable PC with this CPU clearly will require thermal mitigation.

5.3 Adjusting Thermal Headroom for Board Power

Experiments have shown that the term $R \times P_b$ provides a good way to model the effects on the CPU T_C due to other heat sources inside a notebook. (Here R is an experimentally determined thermal coupling coeffi-

cient, and P_b is the board power, as defined in Section 4.). The term adds to T_C, or reduces the $T_A(max)$ that is required to ensure that T_C does not exceed $T_C(max)$:

$$T_A(max) = T_C(max) - \theta_{CA} \times P_{CPU} - R \times P_b$$

To calculate thermal headroom adjusted for the effect of P_b, subtract $R \times P_b$ from the thermal headroom calculated as above. This of course makes the headroom smaller (worse), but by how much? This depends on the size of P_b, but also on R, which is highly dependent on the particular design. In theory, the smallest value possible for R is zero: no thermal coupling between the CPU and other heat sources inside the notebook. From testing one system, the range measured experimentally (with no effort made to thermally isolate the CPU) has been 3.9 to 4.9. Values closer to zero are obtalned by positioning the other high-power devices away from the CPU, and thermally grounding them to the outer case. (Section 5.4 describes how to measure R for a given system.)

5.4 Experimental Measurements are Essential

The θ_{CA} values given by Intel can be used as a rough "rule of thumb" to estimate likely thermal margins. If the thermal headroom calculated from the graphs is positive for a given design even after correcting for the board power ($R = 4$ would be conservative for a real notebook design), the design is most likely satisfactory. But even then, Intel recommends that the CPU T_C be measured when the complete design can be run at full power, with T_A at the maximum allowed by the designer's specifications, to be really sure that $T_C(max)$ will never be exceeded. If a conservative calculation of thermal headroom (including $R \times P_b$) is negative, it is essential that the system be tested, and improvements in thermal mitigation be made until $T_C(max)$ is never exceeded.

There are several levels of thermal experiments that can be used. The simplest is to just measure the CPU T_C, and make adjustments in the design until it never exceeds $T_C(max)$. Then the equations and graphs can be ignored; if the design meets the $T_C(max)$ specification, it does not matter if the (estimated) $T_A(max)$ is violated, as far as Intel's CPU is concerned. (Consideration should be given, however, to other components, such as a disk drive, that might have trouble due to high temperatures inside the notebook.)

Suppose, however, the design is expected to be used for several variations over time, i.e., an Intel486 DX CPU now, and an Intel486 DX2 CPU later, with perhaps higher power peripherals which can also increase P_b in later versions. These later versions with more power will likely require more thermal mitigation efforts, but simply measuring T_C in the first version of the notebook will give little guidance about how much more thermal mitigation will be needed later. In this case, more detailed experiments on the first version, in order to build an accurate thermal model of the product line, can be very cost effective. This can be done using the equation from Section 5.3 and solving for T_C:

$$T_C = T_A + \theta_{CA} \times P_{CPU} + R \times P_b$$

In the preceding equation, T_A is the actual air temperature outside the notebook during the experiment; P_{CPU} is held fixed, and P_b is varied while the resulting T_C is measured. ($T_A + \theta_{CA} \times P_{CPU}$) is the intercept of the resulting straight line, and R is the slope. The easiest way to measure R is to disconnect the CPU (so $P_{CPU} = 0$) and measure T_C with V_{CC} at the upper and lower limits of its range (i.e., 4.5V and 5.5V). P_b is obtained for each V_{CC} value by $V_{CC} \times I_{CC}$. A third data point requires no measurement; when $P_b = 0$, $T_C = T_A$.

When this semi-empirical model has been constructed for a given notebook design, it can be used to accurately determine thermal headroom for variations in both P_{CPU} (plugging in a higher frequency CPU) and P_b (adding higher power peripherals). Of course, if the thermal mechanical design is subsequently modified, R should be measured again for the new version.

If the designers expect the P_b to roughly track P_{CPU}, and R is small (as it should be in a good design), an approximation to the above model may make measurements easier: Assume $P_b = C \times P_{CPU}$, where C is the coefficient relating board power to CPU power. The preceding equation for T_C then becomes:

$$\begin{aligned} T_C &= T_A + \theta_{CA} \times P_{CPU} + R \times P_b \\ &= T_A + \theta_{CA} \times P_{CPU} + R \times C \times P_{CPU} \\ &= T_A + (\theta_{CA} + R \times C)\, P_{CPU} \end{aligned}$$

Then ($\theta_{CA} + R \times C$) becomes a new coefficient, say θ_{CA}', which can be measured by varying P_{CPU} and P_b together by varying V_{CC} over its functional range.

5.5 Secondary Effects

Two kinds of secondary effects will be evaluated. The main model used in thermal analysis assumes a linear relationship between the temperature gradient and the rate of heat transfer, and also assumes a steady state (time independent) model. How valid are these assumptions?

Heat transfer by conduction is governed by a linear relationship between the temperature difference and the rate of heat transfer, and heat transfer by convection can be approximated by a linear relationship, as described in Section 2.1. However, heat transfer by radiation is proportional to the fourth power of the absolute temperatures involved. To demonstrate the contribution that heat transfer by radiation makes to cooling the

CPU, consider the largest allowed T_C (85°C) and the smallest T_A that most designs would find acceptable (40°C). One of the larger CPU packages is 4.4 cm square. Assume the largest emissivity (let $\epsilon = 1$). The temperatures must be converted to degrees Kelvin by adding 273°. Using the formula from Section 2.1, q, the radiative heat transfer in Watts is:

$$q = \epsilon\sigma A (T_C^4 - T_A^4) = 1 * (5.67\ 10^{-8}\ W/(m^2K^4))$$
$$(0.044m)^2 ((358\ K)^4 - (313\ K)^4) = 0.75W$$

To determine the significance of heat transfer by radiation in this case, one compares the 0.75W just calculated to the total heat transfer predicted by the linear approximation using the experimentally determined θ_{CA}. By rearranging the equation

$$\theta_{CA} = \frac{T_C - T_A}{P} \text{ for P, one obtains}$$

$$P = \frac{T_C - T_A}{\theta_{CA}}.$$

$T_C - T_A$ = 45°C in this case, and θ_{CA} ranges 15.5°C/W to 32.0°C/W for the CPUs covered in this article. These figures give a total heat power dissipation ranging from 1.41W to 2.90W. Thus, the radiative component varies between one fourth to one half of the total. This explains why the addition of black paint on a notebook case improved θ_{JA} measurably (see Section 6).

The experimental determination of θ_{CA} effectively incorporates the radiative component, along with the conductive and convective components, in a combined linear approximation, which will be accurate for temperatures near the values used for the measurement. The effect of the radiative, nonlinear component will be beneficial in that the actual power radiated away from the CPU, when temperatures are higher than those used in the θ_{CA} measurements, will be greater than predicted by the linear model. This means that T_C will not increase as much as predicted by the model, for a given increase in power.

The time independent assumption is fine if one is content to build a system to tolerate Case 1 maximum power, and with only passive thermal management (i.e., heat sinks and heat spreaders). However, if one assumes some power averaging over a typical mix of software, as in Cases 2, 3 and 4, one must consider the time dependent effects of bursts of maximum power, alternating with lower power periods. Also, if the design uses active thermal management, especially in a closed loop design with the system responding to a temperature sensor, the lag time between temperature sensing and response must be considered.

The experimental measurements taken on an Intel386 SL CPU notebook show how T_{CASE} varies with time at different CPU power levels, and allow an approximate determination of thermal time constants (see Figure 5-1). The time constant (approximately 3 minutes for this notebook) is defined here as the time elapsed from when power was switched from Full On to Standby, to when the temperature has declined toward its Standby asymptote by 1/e. This means that it is reasonable to average the CPU power over approximately one minute when calculating average power generated by a mix of software. This is a large interval in CPU cycles (billions of CPU clocks).

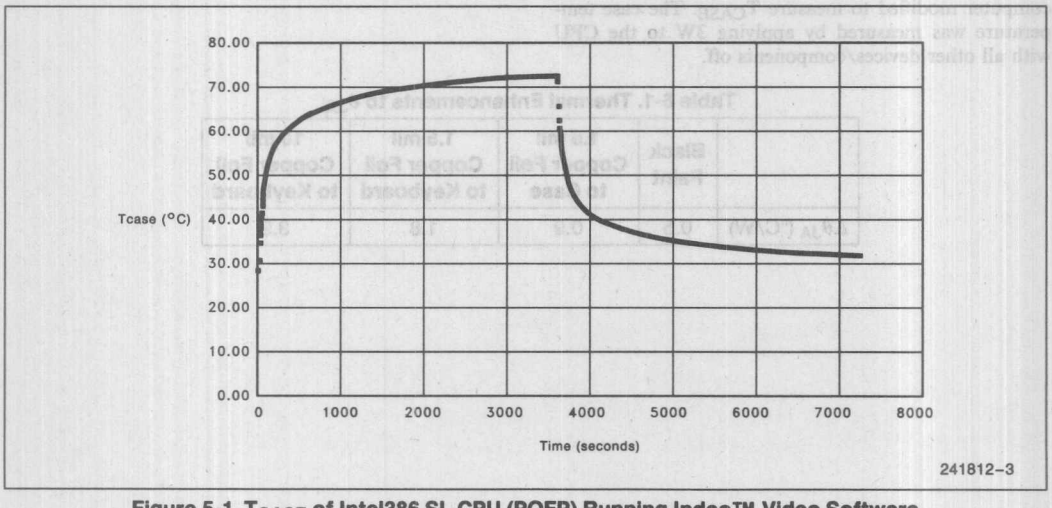


Figure 5-1. T_{CASE} of Intel386 SL CPU (PQFP) Running Indeo™ Video Software

This time constant also indicates a significant lag between a temperature sensor reaching some action value (the action could be turning off the CPU clock for an interval), and a temperature response to that action. This means that the setpoint temperature (the temperature that triggers power reduction) should be somewhat below T_C(max). Note also that the temperature does not come to equilibrium until about an hour after a major power change. This shows that when testing T_C to assure that it does not exceed T_C(max), one should run the notebook under worst case conditions continually for at least an hour.

6.0 IMPROVING THERMAL HEADROOM

By using the thermal management theories that have been reviewed here and keeping in mind the notebook platform limitation, the designer can apply proven thermal management techniques in several areas, including increasing thermal conduction and convection, optimum system layout, and power management techniques.

6.1 Improving Thermal Convection, Conduction and Radiation

The most obvious method for improving thermal convection is by adding a fan to circulate the air. Unfortunately, fans are a compromise in mobile designs because of extra power and space requirements, and electromechanical noise. Intel's Thermal/Mechanical Tools and Analysis Group has collected data using more realistic techniques for reducing T_{CASE}. Data was collected from an actual Intel386 SL microprocessor notebook computer modified to measure T_{CASE}. The case temperature was measured by applying 3W to the CPU with all other devices/components off.

6.1.1 BLACK PAINT

The inside of the notebook case was painted Flat black using a paint that is highly absorptive in the infra-red as well as visible. Figure 6-1 shows that T_{CASE} was improved by 2.3°C, or 3%. Table 6-1 shows that heat transfer improvement by radiation reduces θ_{JA} by 0.5°C/W. The remaining experiments were performed with the inside of the notebook case painted black.

6.1.2 COPPER FOIL

A copper foil (1″ x 3″ x 1.5 mil) was attached from the CPU to the bottom of the keyboard (constructed of aluminum material) and then to the plastic case using thermal grease. Figure 6-1 shows a 4.8°C (6%) and a 2.0°C (3%) improvement when the foil is connected from the CPU to the underside of keyboard and from the CPU to the case, respectively. After connecting the CPU to the bottom of the keyboard, θ_{JA} improved because of the higher thermal dispersion by the aluminum plate. A thicker copper foil (1″ x 3″ x 10 mil) was then connected from the CPU to the keyboard bottom which yielded an improvement of 11.4°C or 15%.

6.1.3 PERFLUOROCARBON FLUID AND SILICONE ELASTOMERS

Liquid heat sinks containing perfluorocarbon fluid can offer a reasonable substitute for standard heat sinks. The heat transfer coefficient for natural convection in a perfluorocarbon fluid is greater than that of natural air convection. Measurements were taken using a perfluorocarbon liquid heat sink connected between the CPU and the plastic case. Figure 6-2 shows a 7.5°C (10%) improvement in temperature.

Table 6-1. Thermal Enhancements to θ_{JA}

	Black Paint	1.5 mil Copper Foil to Case	1.5 mil Copper Foil to Keyboard	10 mil Copper Foil to Keyboard
$\Delta\theta_{JA}$ (°C/W)	0.5	0.9	1.8	3.9

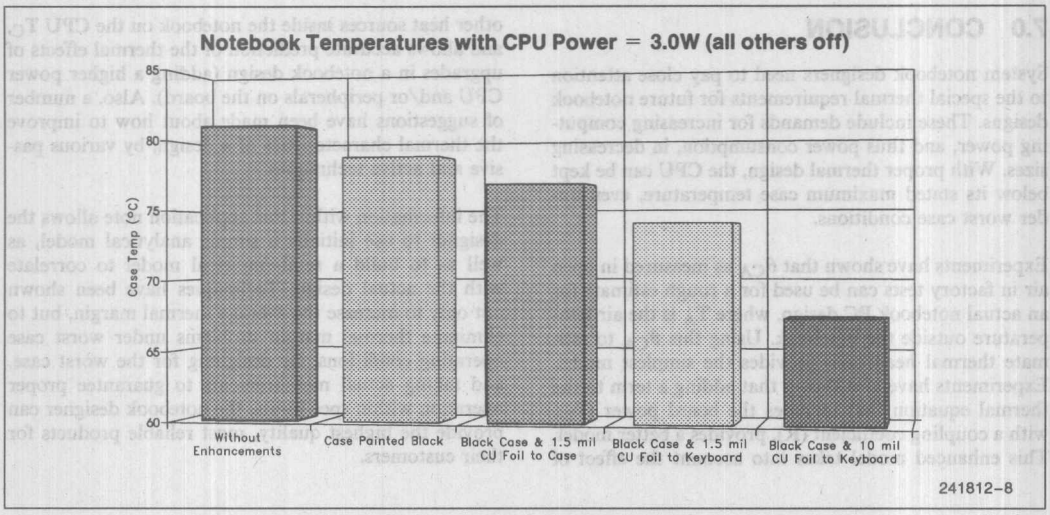


Figure 6-1. Black Paint and Copper Foil Thermal Enhancements

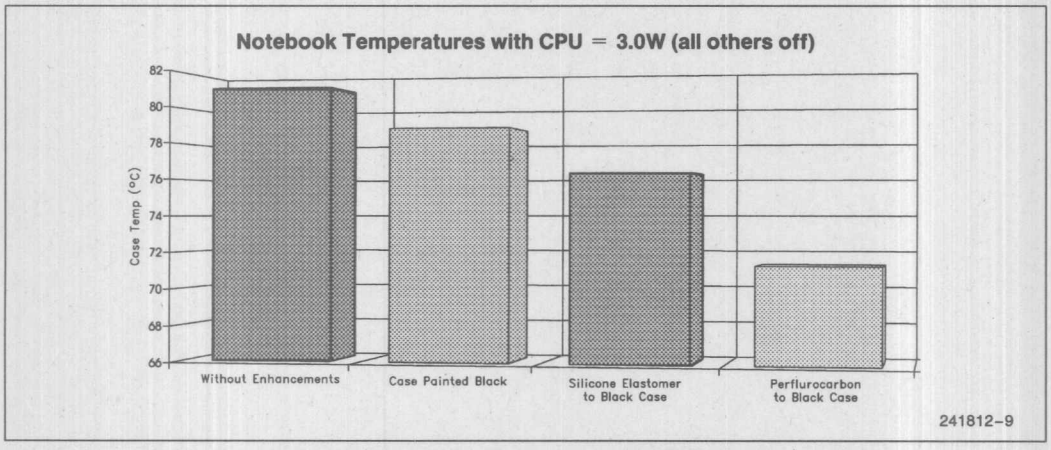


Figure 6-2. Silicone Elastomer and Perfluorocarbon Thermal Enhancements

A type of heat sink that helps blanket uneven surfaces is the silicone elastomers heat sink. These soft materials fill air gaps between hot components and the metal chassis. A piece of silicone elastomer was cut out to the same size as the CPU and placed between the CPU and the case, yielding a 1.5°C (2%) improvement (see Figure 6-2).

6.2 Optimizing System Layout

Power must be optimally distributed to ensure the lowest T_A. The objective is to reduce power density within the system to avoid hot spots at any particular device. Keeping the CPU away from batteries and power supplies is one challenge to the system designer. Thermally connecting the CPU case to the PC case can increase thermal area thus greatly improving thermal spreading. There are many opportunities for creativity in transferring heat away from the CPU.

6.3 Effective System Power Management

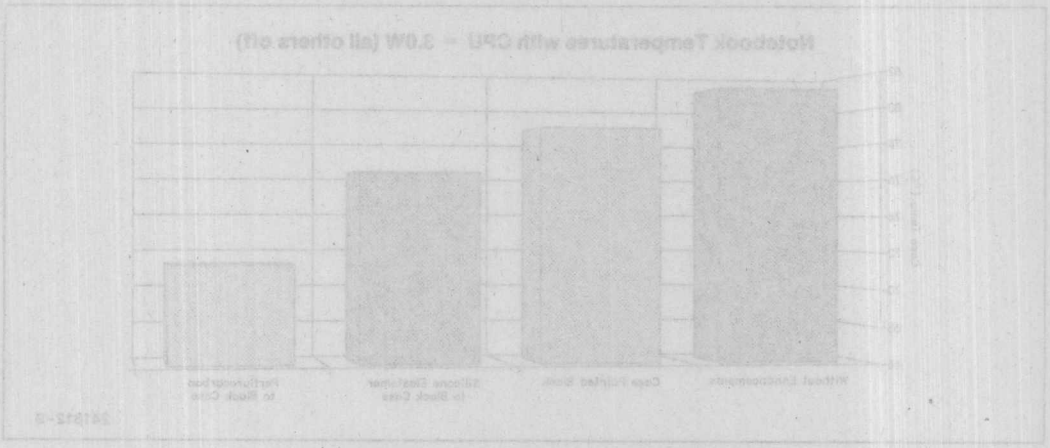
One technique which has become a standard in notebook designs is using Intel's System Management Mode (SMM) to effectively monitor system activity and shut off devices to slow or control clocks when low activity is detected. Another approach could be to monitor the CPU activity or temperature and slow or stop the CPU clock when a long period of system inactivity or a high temperature is detected.

7.0 CONCLUSION

System notebook designers need to pay close attention to the special thermal requirements for future notebook designs. These include demands for increasing computing power, and thus power consumption, in decreasing sizes. With proper thermal design, the CPU can be kept below its stated maximum case temperature, even under worst case conditions.

Experiments have shown that θ_{CA} as measured in open air in factory tests can be used for a rough estimate for an actual notebook PC design, where T_A is the air temperature outside the notebook. Using this θ_{CA} to estimate thermal headroom provides the simplest model. Experiments have also shown that adding a term to the thermal equation that includes the board power (P_b), with a coupling coefficient (R), provides a better model. This enhanced model takes into account the effect of other heat sources inside the notebook on the CPU T_C, and allows accurate prediction of the thermal effects of upgrades in a notebook design (adding a higher power CPU and/or peripherals on the board). Also, a number of suggestions have been made about how to improve the thermal characteristics of a design, by various passive and active techniques.

The information within this application note allows the designer to use initially a simple analytical model, as well as to build a semi-empirical model to correlate with the actual design. Techniques have been shown not only to increase the average thermal margin, but to eliminate thermal margin problems under worst case operating conditions. By designing for the worst case, and taking actual measurements to guarantee proper operation within spec limits, the notebook designer can provide the highest quality, most reliable products for their customers.



APPENDIX A

Table A-1. Power Consumption and Thermal Specifications for SL Enhanced intel486 CPUs

CLK	CPU	V_{CC}	Freq	Pkg	T_{CASE}	V_{CC} tol	I_{CC} Active		I_{CC} Stop Grant		I_{CC} Stop Clock		θ_{JC}	θ_{JA}	θ_{CA} (calc)
							Typ	Max	Typ	Max	Typ	Max			
1X	SX	3.3	25	SQFP	85	±0.30	0.250	0.315	0.020	0.040	0.0001	0.001	4.0	36.0	32.0
		3.3	33	SQFP	85	±0.30	0.300	0.385	0.025	0.050	0.0001	0.001	4.0	36.0	32.0
		5.0	25	PGA	85	±0.25	0.430	0.560	0.035	0.065	0.0002	0.002	1.5	17.0	15.5
		5.0	25	PQFP	85	±0.25	0.430	0.560	0.035	0.065	0.0002	0.002	3.5	20.5	17.0
		5.0	33	PGA	85	±0.25	0.590	0.685	0.040	0.080	0.0002	0.002	1.5	17.0	15.5
		5.0	*33*	*PQFP*	*85*	*±0.25*	*0.590*	*0.685*	*0.040*	*0.080*	*0.0002*	*0.002*	*3.5*	*20.5*	*17.0*
1X	DX	3.3	33	SQFP	85	±0.30	0.330	0.415	0.025	0.050	0.0001	0.001	3.5	25.0	21.5
		5.0	33	PGA	85	+0.25	0.500	0.630	0.040	0.080	0.0002	0.002	1.5	17.0	15.5
		5.0	33	PQFP	85	±0.25	0.500	0.630	0.040	0.080	0.0002	0.002	3.5	20.5	17.0
		5.0	50	PGA	85	±0.25	0.775	0.950	0.050	0.100	0.0002	0.002	1.5	17.0	15.5
1X	DX2	3.3	40	SQFP	85	±0.30	0.375	0.450	0.020	0.040	0.0001	0.001	3.5	24.0	20.5
		3.3	50	SQFP	85	±0.30	0.460	0.550	0.035	0.065	0.0001	0.001	3.5	24.0	20.5
		5.0	50	PGA	85	±0.25	0.775	0.950	0.023	0.050	0.0002	0.002	1.5	17.0	15.5
		5.0	66	PGA	85	±0.25	0.975	1.200	0.045	0.090	0.0002	0.002	1.5	17.0	15.5
2X	SX	3.3	25	SQFP	85	±0.30	0.250	0.315	N/A	N/A	0.0001	0.001	4.0	36.0	32.0
		3.3	33	SQFP	85	±0.30	0.330	0.415			0.0001	0.001	4.0	36.0	32.0
		5.0	25	PQFP	85	±0.25	0.430	0.560			0.0002	0.002	3.5	20.5	17.0
		5.0	33	PQFP	85	±0.25	0.590	0.685			0.0002	0.002	3.5	20.5	17.0
2X	DX	3.3	33	SQFP	85	±0.30	0.330	0.415	N/A	N/A	0.0001	0.001	3.5	25.0	21.5
		5.0	33	PQFP	85	±0.25	0.500	0.630			0.0002	0.002	3.5	20.5	17.0

2

Table A-2. Thermal Headroom based on Typical Power Consumption Profiles of SL Enhanced Intel486 CPUs

CLK	CPU	V_{CC}	Freq	Pkg	T_C	V_{CC} tol	Case 1			Case 2			Case 3			Case 4		
							I_{CC} AVG	Power AVG	Thermal Headrm	I_{CC} AVG	Power AVG	Thermal Headrm	I_{CC} AVG	Power AVG	Thermal Headrm	I_{CC} AVG	Power AVG	Thermal Headrm
1X	SX	3.3	25	SQFP	85	±0.30	0.32	1.04	11.7	0.25	0.83	18.6	0.26	0.85	17.9	0.23	0.76	20.5
		3.3	33	SQFP	85	±0.30	0.39	1.27	4.3	0.30	0.99	13.3	0.31	1.02	12.4	0.28	0.92	15.6
		5.0	25	PGA	85	±0.25	0.56	2.80	1.6	0.43	2.15	11.7	0.44	2.22	10.7	0.40	2.00	14.0
		5.0	25	PQFP	85	±0.25	0.56	2.80	(2.6)	0.43	2.15	8.5	0.44	2.22	7.3	0.40	2.00	11.0
		5.0	33	PGA	85	±0.25	0.69	3.43	(8.1)	0.59	2.95	(0.7)	0.60	3.00	(1.5)	0.54	2.70	3.1
		5.0	*33*	*PQFP*	*85*	*±0.25*	*0.69*	*3.43*	*(13.2)*	*0.59*	*2.95*	*(5.2)*	*0.60*	*3.00*	*(6.0)*	*0.54*	*2.70*	*(1.0)*
1X	DX	3.3	33	SQFP	85	±0.30	0.42	1.37	15.6	0.33	1.09	21.6	0.34	1.12	21.0	0.31	1.01	23.3
		5.0	33	PGA	85	±0.25	0.63	3.15	(3.8)	0.50	2.50	6.3	0.51	2.57	5.2	0.46	2.32	9.1
		5.0	33	PQFP	85	±0.25	0.63	3.15	(8.6)	0.50	2.50	2.5	0.51	2.57	1.4	0.46	2.32	5.6
		5.0	50	PGA	85	±0.25	0.95	4.75	(28.6)	0.78	3.88	(15.1)	0.79	3.96	(16.4)	0.72	3.58	(10.4)
1X	DX2	3.3	40	SQFP	85	±0.30	0.45	1.49	14.6	0.38	1.24	19.6	0.38	1.26	19.1	0.35	1.14	21.7
		3.3	50	SQFP	85	±0.30	0.55	1.82	7.8	0.46	1.52	13.9	0.47	1.55	13.3	0.42	1.40	16.4
		5.0	50	PGA	85	±0.25	0.95	4.75	(28.6)	0.78	3.88	(15.1)	0.79	3.96	(16.4)	0.72	3.58	(10.4)
		5.0	66	PGA	85	±0.25	1.20	6.00	(48.0)	0.98	4.88	(30.6)	1.00	4.99	(32.3)	0.90	4.50	(24.8)
2X	SX	3.3	25	SQFP	85	±0.30	0.32	1.04	11.7	0.25	0.83	18.6	0.26	0.85	17.9	0.23	0.76	20.5
		3.3	33	SQFP	85	±0.30	0.42	1.37	1.2	0.33	1.09	10.2	0.34	1.12	9.3	0.31	1.01	12.7
		5.0	25	PQFP	85	±0.25	0.56	2.80	(2.6)	0.43	2.15	8.5	0.44	2.22	7.3	0.40	2.00	11.0
		5.0	33	PQFP	85	±0.25	0.69	3.43	(13.2)	0.59	2.95	(5.2)	0.60	3.00	(6.0)	0.54	2.70	(1.0)
2X	DX	3.3	33	SQFP	85	±0.30	0.42	1.37	15.6	0.33	1.09	21.6	0.34	1.12	21.0	0.31	1.01	23.3
		5.0	33	PQFP	85	±0.25	0.63	3.15	(8.6)	0.50	2.54	2.5	0.51	2.57	1.4	0.46	2.32	5.6

NOTES:
Case temperature specifications assume a heat spreader and no heat sink.

CASE 1: I_{CC} Active (max) = 100%
CASE 2: I_{CC} Active (typ) = 100%
CASE 3: I_{CC} Active (max) = 10%
I_{CC} Active (typ) = 90%
CASE 4: I_{CC} Active (max) = 10%
I_{CC} Active (typ) = 80%
I_{CC} Stop Clock (max) = 10%

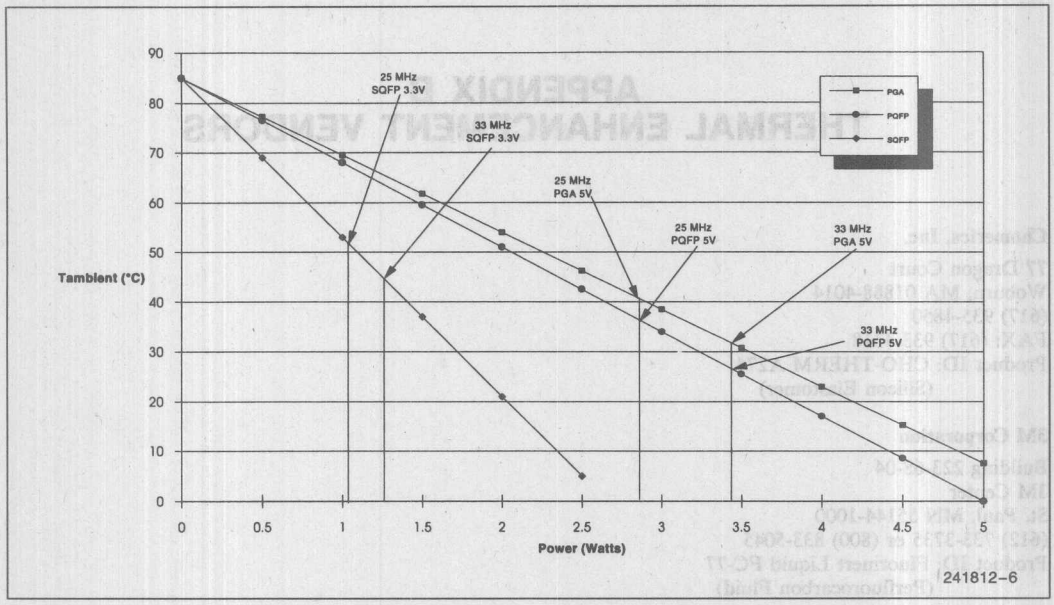


Figure A-1. Maximum Thermal Headroom for SL Enhanced Intel486 SX CPUs

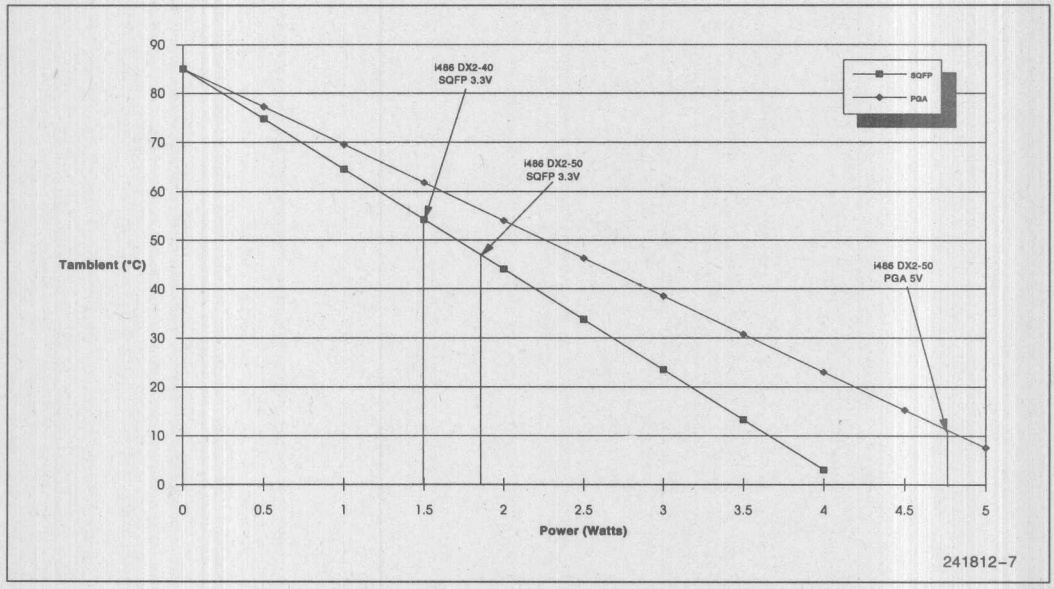


Figure A-2. Maximum Thermal Headroom for SL Enhanced Intel486 DX2 CPUs

APPENDIX B
THERMAL ENHANCEMENT VENDORS

Chomerics, Inc.

77 Dragon Court
Woburn, MA 01888-4014
(617) 935-4850
FAX: (617) 933-4318
Product ID: CHO-THERM A274
(Silicon Elastomer)

3M Corporation

Building 223-6S-04
3M Center
St. Paul, MN 55144-1000
(612) 733-3735 or (800) 833-5045
Product ID: Fluorinert Liquid FC-77
(Perfluorocarbon Fluid)

APPENDIX C
BIBLIOGRAPHY

BIBLIOGRAPHY

SL Enhanced Intel486™ Microprocessor Data Sheet Addendum, Intel Corporation, 1993. Order Number 241696.

Intel386™ SL Microprocessor SuperSet Data Book, Intel Corporation, 1992. Order Number 240814.

1993 Packaging Handbook, Intel Corporation, 1993. Order Number 240800.

Physics, Second Edition, Paul A. Tiper. Worth Publishers, Inc., 1982, pp. 531, 535.

APPENDIX C
BIBLIOGRAPHY

BIBLIOGRAPHY

32- Extended (Intel480™) Microprocessor Data Sheet. Addendum. Intel Corporation, 1993. Order Number 241659.

Intel386™ SL Microprocessor Super Set Data Book. Intel Corporation, 1992. Order Number 240814.

1993 Packaging Handbook. Intel Corporation, 1993. Order Number 240800.

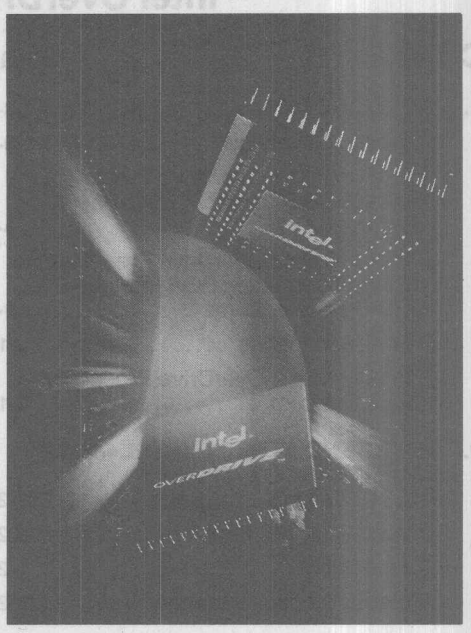
Physics. Second Edition. Paul A. Tipler. Worth Publishers, Inc., 1982, pp. 531, 535.

Intel OverDrive™ Processors

3

INTEL OverDrive™ PROCESSORS

- **Powerful Performance Boosters for Intel486™ Microprocessor-Based Systems**
  - **— Improve Overall System Performance by up to 70%**
  - **— Increase Both Integer and Floating-Point Performance**
  - **— Provides Next Level of CPU Performance**
- **Intel486 DX2 OverDrive Processors Upgrade Systems Based on**
  - **— Intel486™ SX CPUs**
  - **— Intel486™ DX CPUs**
- **Binary Compatible with Large Installed Software Base**
  - **— MS-DOS*, OS/2*, Windows***
  - **— UNIX* System V/386**
  - **— iRMX®, iRMK Kernals**



290436-20

Intel OverDrive processors are powerful, single-chip upgrades for Intel486 microprocessor-based systems. The Intel486 DX2 OverDrive processor upgrades Intel486 SX and DX CPU-based systems with Intel486 DX2's "speed doubling" technology. OverDrive processors accelerate integer and math performance, delivering an overall performance boost of up to 70 percent. Users see a performance boost for all DOS, Windows, OS/2 and UNIX applications from AutoCAD* to WordPerfect*.

*Other brands and names are the property of their respective owners.

October 1993
Order Number: 290436-003

Intel OverDrive™ Processors

CONTENTS

PAGE

1.0 INTRODUCTION 3-3

1.1 Product Overview 3-3

1.2 Intel486 DX2 OverDrive Processor for Intel486 SX and DX CPU-Based Systems 3-4

1.3 Intel486 DX2 OverDrive Processor for Replacement in PGA Intel486 DX CPU-Based Systems 3-8

1.4 Pin Descriptions 3-12

1.5 Intel486 DX2 OverDrive Processor Block Diagram 3-19

2.0 DIFFERENCES FROM INTEL486 SX AND INTEL486 DX CPU FUNCTIONALITY 3-20

2.1 Hardware Interface 3-20

2.2 Testability 3-20

2.3 Instruction Set Summary 3-20

3.0 Intel OverDrive™ PROCESSOR CIRCUIT DESIGN 3-22

3.1 Upgrade Circuit for PGA Intel486 DX CPU-Based Systems 3-22

3.2 Upgrade Circuit for PGA Intel486 SX CPU-Based Systems 3-23

3.3 Upgrade Circuit for PQFP Intel486 SX CPU-Based Systems 3-24

4.0 BIOS AND SOFTWARE 3-25

4.1 Intel OverDrive™ Processor Detection 3-25

4.2 Timing Dependent Loops 3-25

CONTENTS

PAGE

5.0 ELECTRICAL DATA 3-26

5.1 Power and Grounding 3-26

5.2 Maximum Ratings 3-26

5.3 D.C. Specifications 3-27

5.4 A.C. Specifications 3-27

6.0 MECHANICAL DATA 3-36

6.1 Package Dimensions 3-36

6.2 Heat Sink Dimensions 3-37

7.0 THERMAL MANAGEMENT 3-38

7.1 The Intel OverDrive™ Processor with Attached Heat Sink 3-38

7.2 The Intel OverDrive™ Processor without Heat Sink 3-39

7.3 Thermal Equations 3-39

APPENDIX A:

"END USER EASY" UPGRADABILITY 3-41

APPENDIX B:

ZIF AND LIF SOCKET VENDORS 3-42

1.0 INTRODUCTION

This data sheet describes the Intel486 DX2 OverDrive processor. The Intel486 DX2 OverDrive processor is designed as an end-user upgrade for Intel486 SX and Intel486 DX CPU-based systems. This data sheet is intended to be used with the data sheet for the original CPU in the system—the Intel486 SX or Intel486 DX CPU—which describes the Intel486 Family Architecture and functionality. All enhancements or differences in the Intel486 DX2 OverDrive processor from the Intel486 SX or Intel486 DX CPU are described in this data sheet. Intel486 SX or Intel486 DX CPU-based systems that are compatible to the Intel OverDrive processor must be designed to both the original CPU specifications and the Intel OverDrive processor specifications.

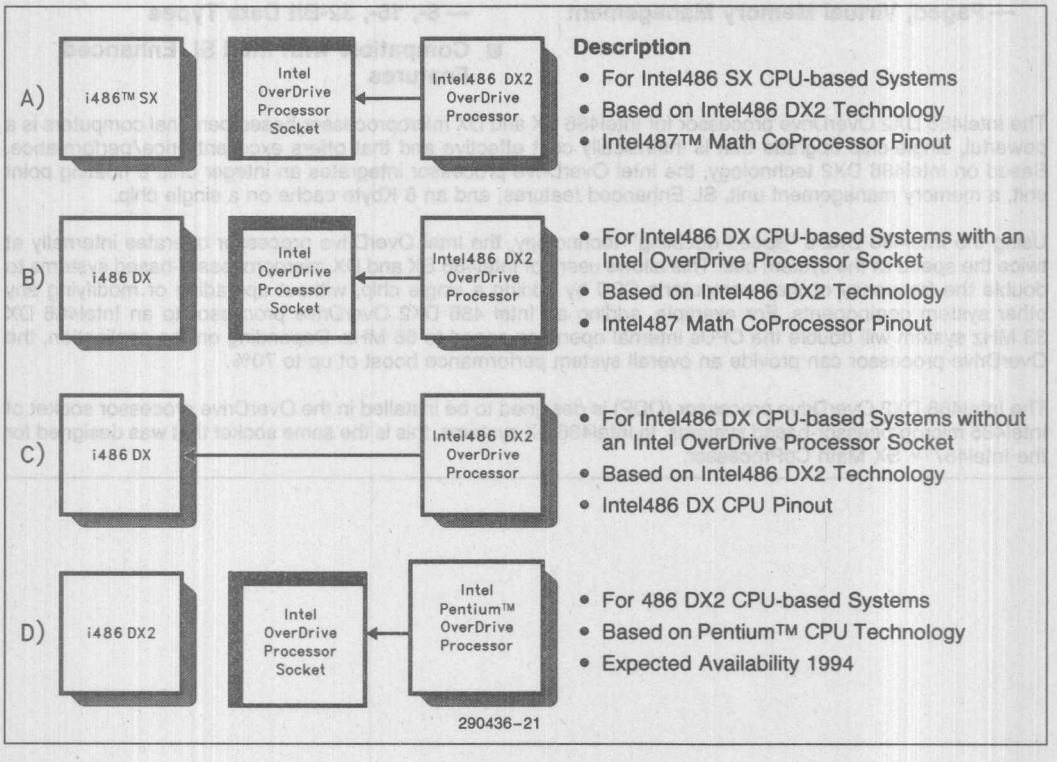
1.1 Product Overview

The Intel486 DX2 OverDrive processor is based on Intel486 DX2 Microprocessor technology. Intel OverDrive processor are designed as a single-chip, powerful performance booster for Intel486 Microprocessor-based systems.

The Intel486 DX2 OverDrive processor is currently available in two basic product variants. The first product variant (ODP), shown in Figures 1A and 1B, is referred to as the OverDrive processor for Intel486 SX and DX CPU-based systems that have an OverDrive processor socket. This product, ODP, is described specifically in Section 1.2.

The second i486 DX2 OverDrive processor variant (ODPR), shown in Figure 1C, is referred to as the OverDrive processor for Intel486 DX CPU-based systems that do not have an OverDrive processor socket. This product is designed to replace the Intel486 DX CPU in systems that do not have an "End-User Easy" OverDrive processor socket. Section 1.3 describes this product variant, ODPR.

3



290436-21

1.2 Intel486 DX2 OverDrive™ Processor for Intel486™ SX and DX Microprocessor-Based Systems (with an OverDrive™ Processor Socket)

- **Powerful Performance Booster for Intel486™ SX and DX CPU-Based Systems**
  - **— Improves Overall System Performance by up to 70%**
  - **— Increases Both Integer and Floating-Point Performance**
- **169-Lead Pin Grid Array Package**
  - **— Pin Compatible with Intel487™ SX Math CoProcessor**
  - **— "End-User Easy", Single-Chip Upgrade**
  - **— 169th Alignment Pin Ensures Proper Chip Orientation**
- **Math CoProcessor Included On-Chip**
- **High Integration Enables On-Chip**
  - **— 8 Kbyte Code and Data Cache**
  - **— Floating Point Unit**
  - **— Paged, Virtual Memory Management**
- **Utilizes Intel486™ DX2 Speed-Doubling Technology**
  - **— CPU Core Runs at Twice the Frequency of the System Bus**
  - **— Compatible with 33, 25, 20, and 16 MHz Systems**
- **Binary Compatible with Large Installed Software Base**
  - **— MS-DOS, OS/2, Windows**
  - **— UNIX System V/386**
  - **— iRMX®, iRMK Kernals**
- **High Performance Design**
  - **— Core Clock Speed up to 66 MHz**
  - **— 106 Mbyte/sec Burst Bus**
  - **— CHMOS V Process Technology**
- **Complete 32-Bit Architecture**
  - **— Address and Data Busses**
  - **— Registers**
  - **— 8-, 16-, 32-Bit Data Types**
- **Compatible with Intel SL Enhanced Features**

The Intel486 DX2 OverDrive processor for Intel486 SX and DX microprocessor-based personal computers is a powerful, single-chip upgrade that is intrinsically cost effective and that offers excellent price/performance. Based on Intel486 DX2 technology, the Intel OverDrive processor integrates an integer unit, a floating point unit, a memory management unit, SL Enhanced features, and an 8 Kbyte cache on a single chip.

Using the Intel486 DX2's "speed doubling" technology, the Intel OverDrive processor operates internally at twice the speed of the system bus. This allows users of Intel486 SX and DX microprocessor-based systems to double the frequency of their computer's CPU by adding a single chip, without upgrading or modifying any other system components. For example, adding an Intel 486 DX2 OverDrive processor to an Intel486 DX 33 MHz system will double the CPUs internal operating speed to 66 MHz. Depending on the application, the OverDrive processor can provide an overall system performance boost of up to 70%.

The Intel486 DX2 OverDrive processor (ODP) is designed to be installed in the OverDrive processor socket of Intel486 microprocessor-based systems. In Intel486 SX systems, this is the same socket that was designed for the Intel487™ SX Math CoProcessor.

The Intel486 DX2 OverDrive processor is available in three product versions. The 20 MHz Intel OverDrive processor is designed to upgrade both 20 MHz and 16 MHz Intel486 SX Microprocessor-based systems. This product utilizes the standard 169-lead PGA package. The 25 MHz and 33 MHz Intel OverDrive processor is designed to upgrade 25 MHz and 33 MHz Intel486 SX or DX Microprocessor-based systems. This product has a heat sink attached to the standard 169-lead PGA package to aid in the heat dissipation in 25 MHz and 33 MHz systems.

1.2.1 Pinout

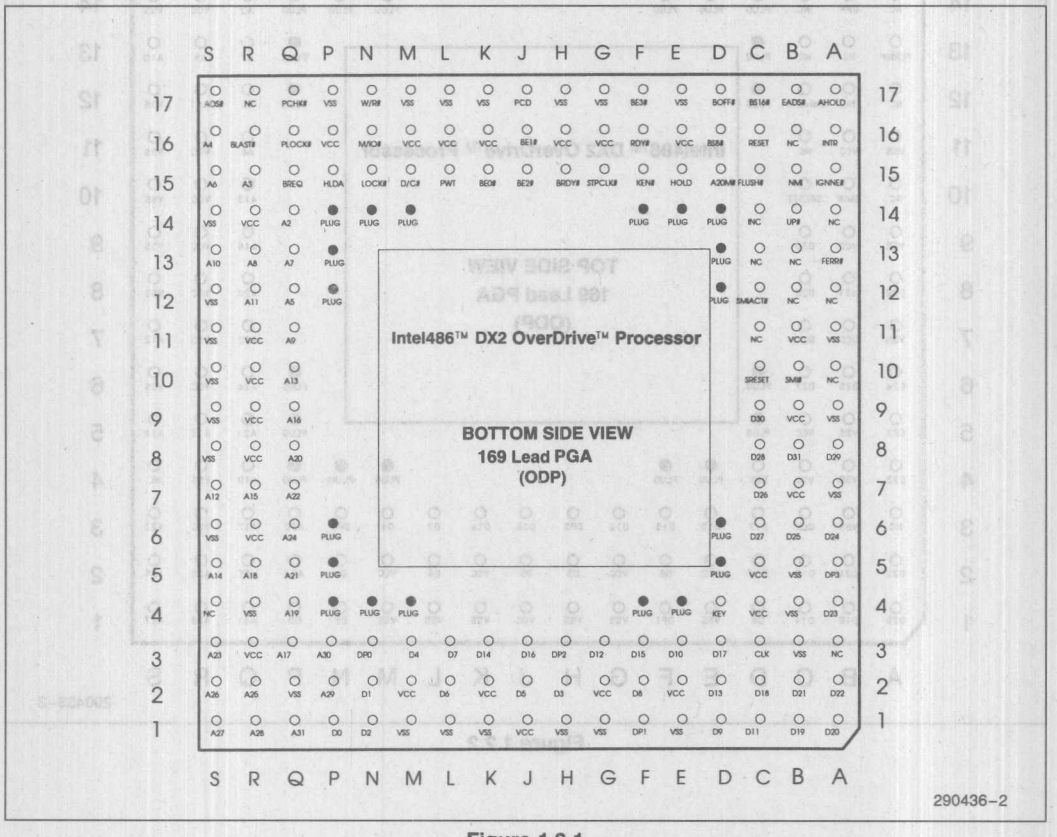


Figure 1.2.1

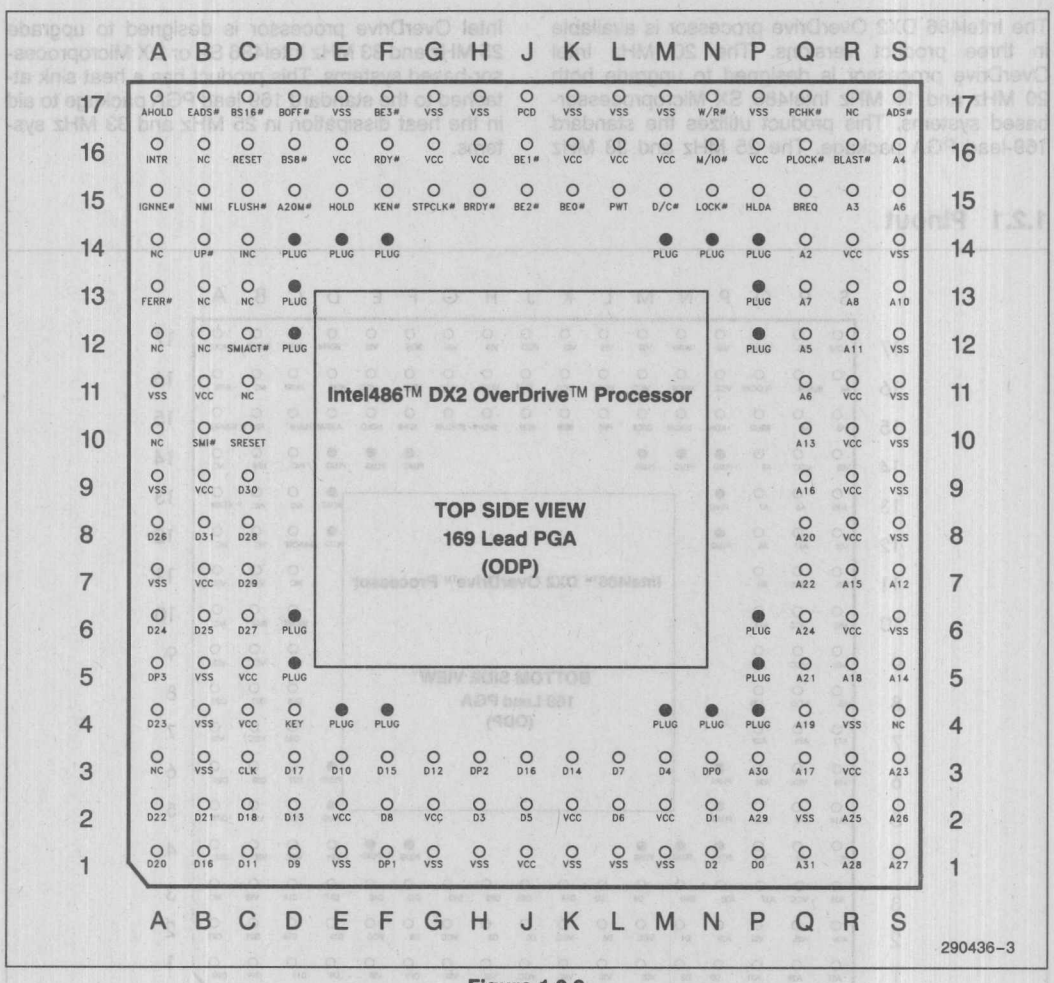

A B C D E F G H J K L M N P Q R S
17 AHOLD EADS# BS16# BOFF# VSS BE3# VSS VSS PCD VSS VSS VSS W/R# VSS PCHK# NC ADS#
16 INTR NC RESET BS8# VCC RDY# VCC VCC BE1# VCC VCC VCC M/IO# VCC PLOCK# BLAST# A4
15 IGNNE# NMI FLUSH# A20M# HOLD KEN# STPCLK# BRDY# BE2# BE0# PWT D/C# LOCK# HLDA BREQ A3 A6
14 NC UP# INC PLUG PLUG PLUG PLUG PLUG PLUG PLUG A2 VCC VSS
13 FERR# NC NC PLUG PLUG A7 A8 A10
12 NC NC SMIACT# PLUG PLUG A5 A11 VSS
11 VSS VCC NC A6 VCC VSS
10 NC SMI# SRESET A13 VCC VSS
9 VSS VCC D30 A16 VCC VSS
8 D26 D31 D28 A20 VCC VSS
7 VSS VCC D29 A22 A15 A12
6 D24 D25 D27 PLUG PLUG A24 VCC VSS
5 DP3 VSS VCC PLUG PLUG A21 A18 A14
4 D23 VSS VCC KEY PLUG PLUG PLUG PLUG PLUG A19 VSS NC
3 NC VSS CLK D17 D10 D15 D12 DP2 D16 D14 D7 D4 DP0 A30 A17 VCC A23
2 D22 D21 D18 D13 VCC D8 VCC D3 D5 VCC D6 VCC D1 A29 VSS A25 A26
1 D20 D16 D11 D9 VSS DP1 VSS VSS VCC VSS VSS VSS D2 D0 A31 A28 A27
A B C D E F G H J K L M N P Q R S
Intel486™ DX2 OverDrive™ Processor
TOP SIDE VIEW
169 Lead PGA
(ODP)
290436-3

Figure 1.2.2

Table 1.2. Pin Cross Reference by Pin Name (ODP)

Address		Data		Control		N/C(1)	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	A10	B7	A7
A_3	R15	D_1	N2	ADS#	S17	A12	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	A14	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	B12	C4	B3
A_6	S15	D_4	M3	BE1#	J16	B13	C5	B4
A_7	Q13	D_5	J2	BE2#	J15	C13	E16	B5
A_8	R13	D_6	L2	BE3#	F17	R17	G16	E1
A_9	Q11	D_7	L3	BLAST#	R16	S4	H16	E17
A_{10}	S13	D_8	F2	BOFF#	D17	A3	J1	G1
A_{11}	R12	D_9	D1	BRDY#	H15	B16	K16	G17
A_{12}	S7	D_{10}	E3	BREQ#	Q15	C11	L16	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16		M2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17		M16	K1
A_{15}	R7	D_{13}	D2	CLK	C3		P16	K17
A_{16}	Q9	D_{14}	K3	D/C#	M15	INC(2)	R3	L1
A_{17}	Q3	D_{15}	F3	DP0	N3			L17
A_{18}	R5	D_{16}	J3	DP1	F1	C14	R6	M1
A_{19}	Q4	D_{17}	D3	DP2	H3		R8	M17
A_{20}	Q8	D_{18}	C2	DP3	A5		R9	P17
A_{21}	Q5	D_{19}	B1	EADS#	B17	PLUG	R10	Q2
A_{22}	Q7	D_{20}	A1	FERR#	A13			R4
A_{23}	S3	D_{21}	B2	FLUSH#	C15	D5	R11	S6
A_{24}	Q6	D_{22}	A2	HLDA	P15	D6	R14	S8
A_{25}	R2	D_{23}	A4	HOLD	E15	D12		S9
A_{26}	S2	D_{24}	A6	IGNNE#	A15	D13		S10
A_{27}	S1	D_{25}	B6	INTR	A16	D14		S11
A_{28}	R1	D_{26}	C7	KEN#	F15	E4		S12
A_{29}	P2	D_{27}	C6	LOCK#	N15	E14		S14
A_{30}	P3	D_{28}	C8	M/IO#	N16	F4		
A_{31}	Q1	D_{29}	A8	NMI	B15	F14		
		D_{30}	C9	PCD	J17	M4		
		D_{31}	B8	PCHK#	Q17	M14		
				PWT	L15	N4		
				PLOCK#	Q16	N14		
				RDY#	F16	P4		
				RESET	C16	P5		
				SMI#	B10	P6		
				SMIACT#	C12	P12		
				SRESET	C10	P13		
				STPCLK#	G15	P14		
				UP#	B14			
				W/R#	N17			
				KEY	D4			

NOTES:
1. All NC pins must remain unconnected.
2. The INC pin is defined to be an internal no-connect. This means that the pin is not internally connected and may be used for the routing of external signals.

1.3 Intel486 DX2 OverDrive™ Processor for PGA Intel486™ DX Microprocessor-Based Systems (without an OverDrive™ Processor Socket)

- **Powerful Performance Booster for PGA Intel486™ DX CPU-Based Systems**
  - **— Improves Overall System Performance by up to 70%**
  - **— Increases Both Integer and Floating-Point Performance**
- **168-Lead Pin Grid Array Package**
  - **— Pin Compatible with Intel486™ DX CPU**
- **Math CoProcessor Included On-Chip**
- **High Integration Enables On-Chip**
  - **— 8 Kbyte Code and Data Cache**
  - **— Floating Point Unit**
  - **— Paged, Virtual Memory Management**
- **Compatible with SL Enhanced Features**
- **Utilizes Intel486™ DX2 Speed-Doubling Technology**
  - **— CPU Core Runs at Twice the Frequency of the System Bus**
  - **— Compatible with 33 and 25 MHz Systems**
- **Binary Compatible with Large Installed Software Base**
  - **— MS-DOS, OS/2, Windows**
  - **— UNIX System V/386**
  - **— iRMX®, iRMK Kernals**
- **High Performance Design**
  - **— Core Clock Speed up to 66 MHz**
  - **— 106 Mbyte/sec Burst Bus**
  - **— CHMOS V Process Technology**
- **Complete 32-Bit Architecture**
  - **— Address and Data Busses**
  - **— Registers**
  - **— 8-, 16-, 32-Bit Data Types**

The Intel486 DX2 OverDrive processor for Intel486 DX microprocessor-based personal computers is a powerful, single-chip upgrade that is intrinsically cost effective and that offers excellent price/performance. Based on Intel486 DX2 technology, the Intel OverDrive processor integrates an integer unit, a floating point unit, a memory management unit and an 8 Kbyte cache on a single chip.

Using the Intel486 DX2's "speed doubling" technology, the Intel OverDrive processor operates internally at twice the speed of the system bus. This allows users of Intel486 DX microprocessor-based systems to double the frequency of their computer's CPU by replacing a single chip, without upgrading or modifying any other system components. For example, adding an Intel486 DX2 OverDrive processor to an Intel486 DX 33 MHz system will double the CPUs internal operating speed to 66 MHz. Depending on the application, the OverDrive processor can provide an overall system performance boost of up to 70%.

The Intel486 DX2 OverDrive processor, ODPR, is designed to be installed in the Intel486 DX CPU socket after the Intel486 DX CPU has been removed.

Both the 25 MHz and 33 MHz versions of the OverDrive processor for Intel486 DX CPU-based Systems without an OverDrive processor socket have a heat sink attached to the standard 168-lead PGA package.

1.3.1 Pinout

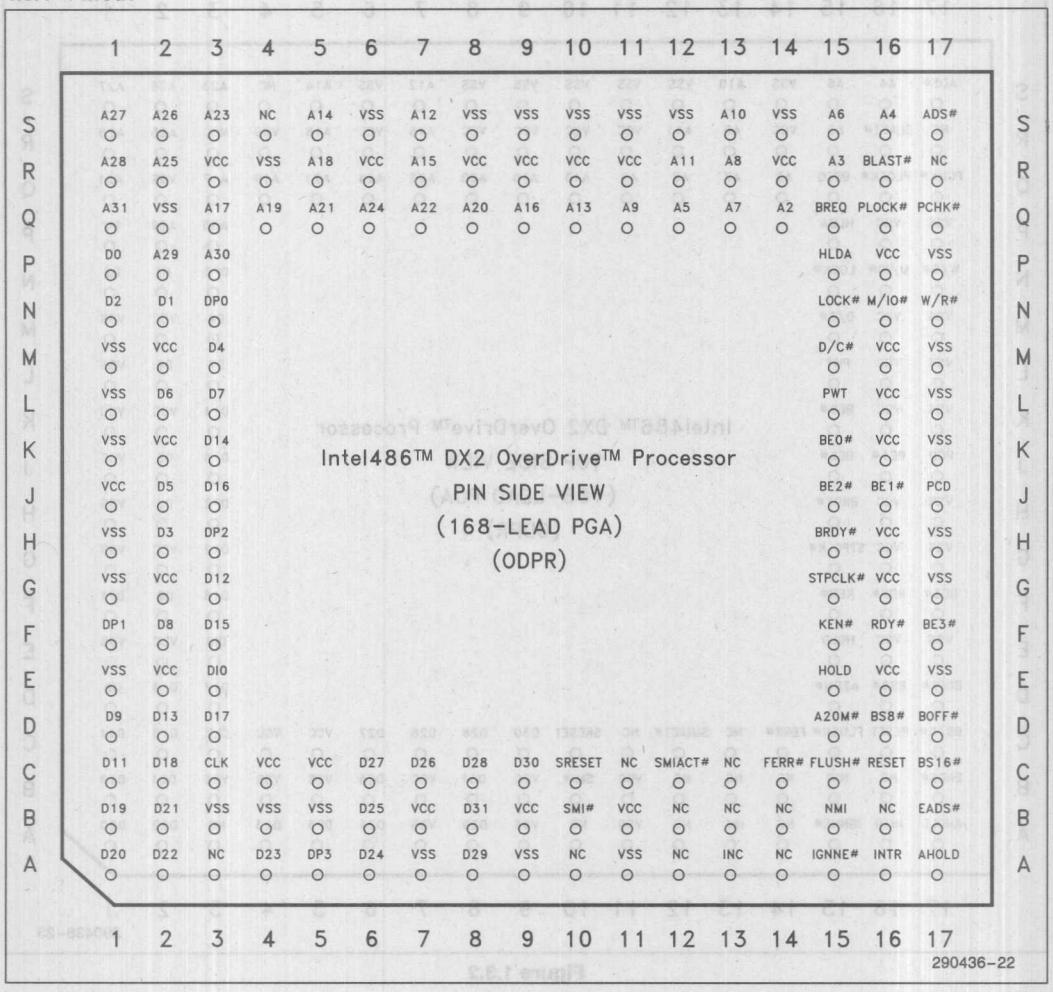


Figure 1.3.1

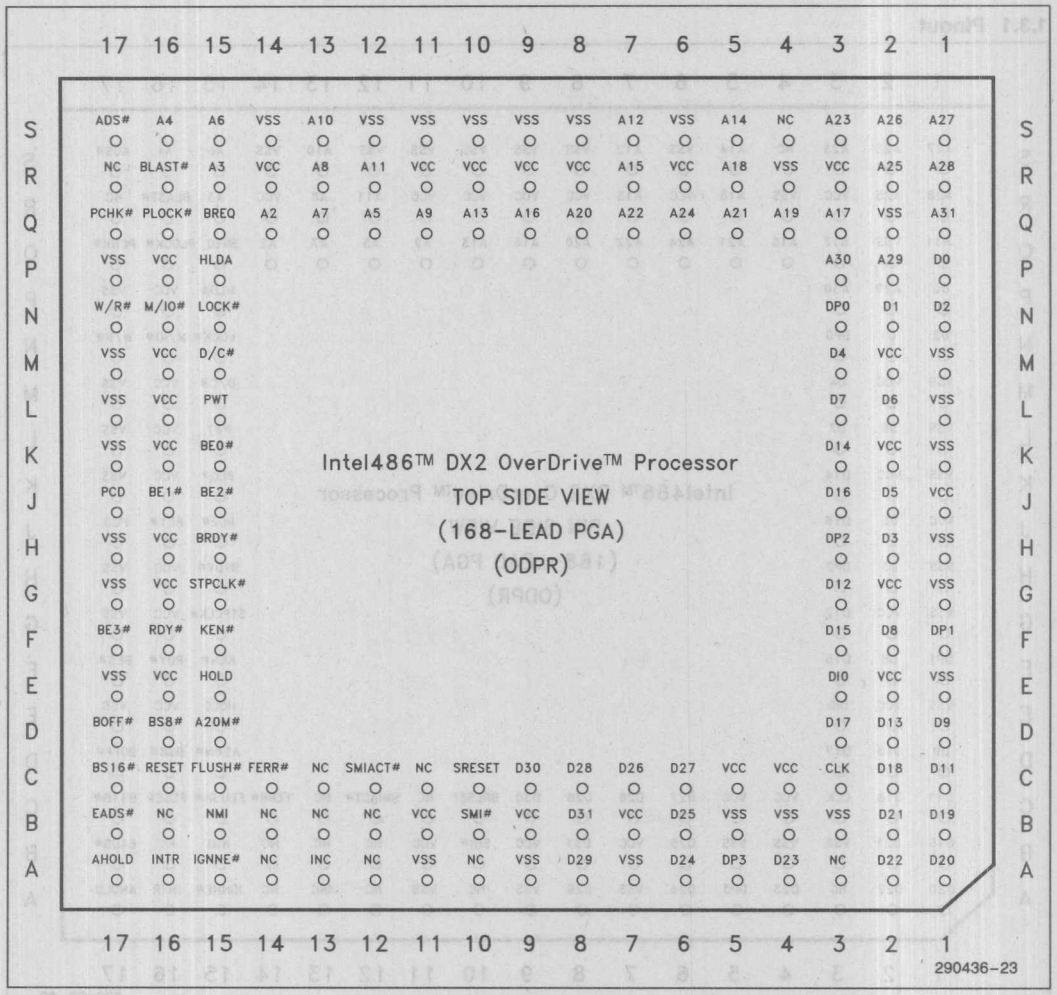

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
S ADS# A4 A6 VSS A10 VSS VSS VSS VSS VSS A12 VSS A14 NC A23 A26 A27
R NC BLAST# A3 VCC A8 A11 VCC VCC VCC VCC A15 VCC A18 VSS VCC A25 A28
Q PCHK# PLOCK# BREQ A2 A7 A5 A9 A13 A16 A20 A22 A24 A21 A19 A17 VSS A31
P VSS VCC HLDA A30 A29 D0
N W/R# M/IO# LOCK# DP0 D1 D2
M VSS VCC D/C# D4 VCC VSS
L VSS VCC PWT D7 D6 VSS
K VSS VCC BE0# D14 VCC VSS
J PCD BE1# BE2# D16 D5 VCC
H VSS VCC BRDY# DP2 D3 VSS
G VSS VCC STPCLK# D12 VCC VSS
F BE3# RDY# KEN# D15 D8 DP1
E VSS VCC HOLD DI0 VCC VSS
D BOFF# BS8# A20M# D17 D13 D9
C BS16# RESET FLUSH# FERR# NC SMIACT# NC SRESET D30 D28 D26 D27 VCC VCC CLK D18 D11
B EADS# NC NMI NC NC NC VCC SMI# VCC D31 VCC D25 VSS VSS VSS D21 D19
A AHOLD INTR IGNNE# NC INC NC VSS NC VSS D29 VSS D24 DP3 D23 NC D22 D20
Intel486™ DX2 OverDrive™ Processor
TOP SIDE VIEW
(168-LEAD PGA)
(ODPR)
17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
290436-23

Figure 1.3.2

Table 1.3. Pin Cross Reference by Pin Name (ODPR)

Address		Data		Control		N/C(1)	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	A3	B7	A7
A_3	R15	D_1	N2	ADS#	S17	A10	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	A12	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	A14	C4	B3
A_6	S15	D_4	M3	BE1#	J16	B12	C5	B4
A_7	Q13	D_5	J2	BE2#	J15	B13	E2	B5
A_8	R13	D_6	L2	BE3#	F17	B14	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16	B16	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17	C11	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15	C13	H16	G17
A_{12}	S7	D_{10}	E3	BREQ#	Q15	R17	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16	S4	K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17		K16	K1
A_{15}	R7	D_{13}	D2	CLK	C3		L16	K17
A_{16}	Q9	D_{14}	K3	D/C#	M15	INC(2)	M2	L1
A_{17}	Q3	D_{15}	F3	DP0	N3		M16	L17
A_{18}	R5	D_{16}	J3	DP1	F1	A13	P16	M1
A_{19}	Q4	D_{17}	D3	DP2	H3		R3	M17
A_{20}	Q8	D_{18}	C2	DP3	A5		R6	P17
A_{21}	Q5	D_{19}	B1	EADS#	B17		R8	Q2
A_{22}	Q7	D_{20}	A1	FERR#	C14		R9	R4
A_{23}	S3	D_{21}	B2	FLUSH#	C15		R10	S6
A_{24}	Q6	D_{22}	A2	HLDA	P15		R11	S8
A_{25}	R2	D_{23}	A4	HOLD	E15		R14	S9
A_{26}	S2	D_{24}	A6	IGNNE#	A15			S10
A_{27}	S1	D_{25}	B6	INTR	A16			S11
A_{28}	R1	D_{26}	C7	KEN#	F15			S12
A_{29}	P2	D_{27}	C6	LOCK#	N15			S14
A_{30}	P3	D_{28}	C8	M/IO#	N16			
A_{31}	Q1	D_{29}	A8	NMI	B15			
		D_{30}	C9	PCD	J17			
		D_{31}	B8	PCHK#	Q17			
				PWT	L15			
				PLOCK#	Q16			
				RDY#	F16			
				RESET	C16			
				SMI#	B10			
				SMIACT#	C12			
				SRESET	C10			
				STPCLK#	G15			
				W/R#	N17			

NOTES:
1. All NC pins must remain unconnected.
2. The INC pin is defined to be internal no-connect. This means that the pin is not internally connected and may be used for the routing of external signals.

1.4 PIN DESCRIPTIONS

What follows is a brief pin description.

Symbol	Type	Name and Function
CLK	I	*Clock* provides the fundamental timing for the bus interface unit and is multiplied by two (2x) to provide the internal frequency for the Intel OverDrive processor. All external timing parameters are specified with respect to the rising edge of CLK.
ADDRESS BUS		
A31–A4 A2–A3	I/O O	A31–A2 are the *address lines* of the processor. A31–A2, together with the byte enables BE0#–BE3#, define the physical area of memory or input/output space accessed. Address lines A31–A4 are used to drive addresses into the processor to perform cache line invalidations. Input signals must meet setup and hold times t_{22} and t_{23}. A31–A2 are not driven during bus or address hold.
BE0–3#	O	The *byte enable* signals indicate active bytes during read and write cycles. During the first cycle of a cache fill, the external system should assume that all byte enables are active. BE3# applies to D24–D31, BE2# applies to D16–D23, BE1# applies to D8–D15 and BE0# applies to D0–D7. BE0#–BE3# are active LOW and are not driven during bus hold.
DATA BUS		
D31–D0	I/O	These are the *data lines* for the Intel OverDrive processor. Lines D0–D7 define the least significant byte of the data bus while lines D24–D31 define the most significant byte of the data bus. These signals must meet setup and hold times t_{22} and t_{23} for proper operation on reads. These pins are driven during the second and subsequent clocks of write cycles.
DATA PARITY		
DP0–DP3	I/O	There is one *data parity* pin for each byte of the data bus. Data parity is generated on all write data cycles with the same timing as the data driven by the Intel OverDrive processor. Even parity information must be driven back into the microprocessor on the data parity pins with the same timing as read information to insure that the correct parity check status is indicated by the Intel OverDrive processor. The signals read on these pins do not affect program execution. Input signals must meet setup and hold times t_{22} and t_{23}. DP0–DP3 should be connected to V_{CC} through a pullup resistor in systems which do not use parity. DP0–DP3 are active HIGH and are driven during the second and subsequent clocks of write cycles.
PCHK#	O	*Parity Status* is driven on the PCHK# pin the clock after ready for read operations. The parity status is for data sampled at the end of the previous clock. A parity error is indicated by PCHK# being LOW. Parity status is only checked for enabled bytes as indicated by the byte enable and bus size signals. PCHK# is valid only in the clock immediately after read data is returned to the microprocessor. At all other times PCHK# is inactive (HIGH). PCHK# is never floated.

1.4 PIN DESCRIPTIONS (Continued)

Symbol	Type	Name and Function
BUS CYCLE DEFINITION		
M/IO# D/C# W/R#	O O O	The *memory/input-output, data/control* and *write/read* lines are the primary bus definition signals. These signals are driven valid as the ADS# signal is asserted.

M/IO#	D/C#	W/R#	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Halt/Special Cycle
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Code Read
1	0	1	Reserved
1	1	0	Memory Read
1	1	1	Memory Write

The bus definition signals are not driven during bus hold and follow the timing of the address bus. Refer to Section 7.2.11 for a description of the special bus cycles.

LOCK#	O	The *bus lock* pin indicates that the current bus cycle is locked. The Intel OverDrive processor will not allow a bus hold when LOCK# is asserted (but address holds are allowed). LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after the last clock of the last locked bus cycle. The last locked cycle ends when RDY# is returned. LOCK# is active LOW and is not driven during bus hold. Locked read cycles will not be transformed into cache fill cycles if KEN# is returned active.
PLOCK#	O	The *pseudo-lock* pin indicates that the current bus transaction requires more than one bus cycle to complete. Examples of such operations are floating point long reads and writes (64 bits), segment table descriptor reads (64 bits), in addition to cache line fills (128 bits). The Intel OverDrive processor will drive PLOCK# active until the addresses for the last bus cycle of the transaction have been driven regardless of whether RDY# or BRDY# have been returned. Normally PLOCK# and BLAST# are inverse of each other. However during the first bus cycle of a 64-bit floating point write, both PLOCK# and BLAST# will be asserted. PLOCK# is a function of the BS8#, BS16# and KEN# inputs. PLOCK# should be sampled only in the clock RDY# is returned. PLOCK# is active LOW and is not driven during bus hold.
BUS CONTROL		
ADS#	O	The *address status* output indicates that a valid bus cycle definition and address are available on the cycle definition lines and address bus. ADS# is driven active in the same clock as the addresses are driven. ADS# is active LOW and is not driven during bus hold.
RDY#	I	The *non-burst ready* input indicates that the current bus cycle is complete. RDY# indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted data from the Intel OverDrive processor in response to a write. RDY# is ignored when the bus is idle and at the end of the first clock of the bus cycle. RDY# is active during address hold. Data can be returned to the processor while AHOLD is active. RDY# is active LOW, and is not provided with an internal pullup resistor. RDY# must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.

1.4 PIN DESCRIPTIONS (Continued)

Symbol	Type	Name and Function
BURST CONTROL		
BRDY#	I	The *burst ready input* performs the same function during a burst cycle that RDY# performs during a non-burst cycle. BRDY# indicates that the external system has presented valid data in response to a read or that the external system has accepted data in response to a write. BRDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle. BRDY# is sampled in the second and subsequent clocks of a burst cycle. The data presented on the data bus will be strobed into the microprocessor when BRDY# is sampled active. If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is prematurely interrupted BRDY# is active LOW and is provided with a small pullup resistor. BRDY# must satisfy the setup and hold times t_{16} and t_{17}.
BLAST#	O	The *burst last* signal indicates that the next time BRDY# is returned the burst bus cycle is complete. BLAST# is active for both burst and non-burst bus cycles. BLAST# is active LOW and is not driven during bus hold.
INTERRUPTS		
RESET	I	The **RESET** input forces the CPU to begin execution at a known state. Reset is asynchronous, but must meet setup and hold times t20 and t21 for recognition in any specific clock. The CPU cannot begin execution of instructions until at least 1 ms after V_{CC} and CLK have reached their proper AC and DC specifications. However, for soft resets, RESET should remain active for at least 15 CLK periods. The RESET pin should remain active during this time to ensure proper CPU operation. RESET is active HIGH. RESET sets the SMBASE descriptor to a default address of 30000H. If the system uses SMBASE relocation, then the SRESET pin should be used for soft resets.
SRESET	I	The SRESET pin duplicates all the functionality of the RESET pin with the following two exceptions: 1. The SMBASE register will retain its previous value. 2. If UP# (I) is asserted, SRESET will not have an effect on the host microprocessor. For soft resets, SRESET should remain active for at least 15 CLK periods. SRESET is active HIGH. SRESET is asynchronous but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
SMI#	I	The **System Management Interrupt** input is used to invoke the System Management Mode (SMM). SMI# is a falling edge triggered signal which forces the CPU into SMM at the completion of the current instruction. SMI# is recognized on an instruction boundary and at each iteration for repeat string instructions. SMI# does not break LOCKed bus cycles and cannot interrupt a currently executing SMM. The CPU will latch the falling edge of one pending SMI# signal while the CPU is executing an existing SMI. The nested SMI will not be recognized until after the execution of a Resume (RSM) instruction.
SMIACT#	O	The **System Management Interrupt ACTive** is an active low output, indicating that the processor is operating in SMM. It is asserted when the CPU begins to execute the SMI state save sequence and will remain active LOW until the processor executes the last state restore cycle out of SMRAM.
STPCLK#	I	The **SToP CLocK request** input signal indicates a request has been made to turn off the CLK input. When the CPU recognizes a STPCLK#, the processor will stop execution on the next instruction boundary, unless superseded by a higher priority interrupt, empty all internal pipelines and the write buffers and generate a Stop Grant acknowledge bus cycle. STPCLK# is active LOW and is provided with an internal pull-up resistor. STPCLK# is asynchronous but setup and hold times t_{20} and t_{21} must be met to ensure recognition in any specific clock.

1.4 PIN DESCRIPTIONS (Continued)

Symbol	Type	Name and Function
INTERRUPTS (Continued)		
INTR	I	The *maskable interrupt* indicates that an external interrupt has been generated. If the internal interrupt flag is set in EFLAGS, active interrupt processing will be initiated. The Intel OverDrive processor will generate two locked interrupt acknowledge bus cycles in response to the INTR pin going active. INTR must remain active until the interrupt acknowledges have been performed to assure that the interrupt is recognized. INTR is active HIGH and is not provided with an internal pulldown resistor. INTR is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
NMI	I	The *non-maskable interrupt* request signal indicates that an external non-maskable interrupt has been generated. NMI is rising edge sensitive. NMI must be held LOW for at least four CLK periods before this rising edge. NMI is not provided with an internal pulldown resistor. NMI is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
BUS ARBITRATION		
BREQ	O	The *internal cycle pending* signal indicates that the Intel OverDrive processor has internally generated a bus request. BREQ is generated whether or not the Intel OverDrive processor is driving the bus. BREQ is active HIGH and is never floated.
HOLD	I	The *bus hold request* allows another bus master complete control of the Intel OverDrive processor bus. In response to HOLD going active the Intel OverDrive processor will float most of its output and input/output pins. HLDA will be asserted after completing the current bus cycle, burst cycle or sequence of locked cycles. The Intel OverDrive processor will remain in this state until HOLD is deasserted. HOLD is active high and is not provided with an internal pulldown resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper operation.
HLDA	O	*Hold acknowledge* goes active in response to a hold request presented on the HOLD pin. HLDA indicates that the Intel OverDrive processor has given the bus to another local bus master. HLDA is driven active in the same clock that the Intel OverDrive processor floats its bus. HLDA is driven inactive when leaving bus hold. HLDA is active HIGH and remains driven during bus hold.

1.4 PIN DESCRIPTIONS (Continued)

Symbol	Type	Name and Function
BUS ARBITRATION (Continued)		
BOFF#	I	The *backoff* input forces the Intel OverDrive processor to float its bus in the next clock. The microprocessor will float all pins normally floated during bus hold but HLDA will not be asserted in response to BOFF#. BOFF# has higher priority than RDY# or BRDY#; if both are returned in the same clock, BOFF# takes effect. The microprocessor remains in bus hold until BOFF# is negated. If a bus cycle was in progress when BOFF# was asserted the cycle will be restarted. BOFF# is active LOW and must meet setup and hold times t_{18} and t_{19} for proper operation.
CACHE INVALIDATION		
AHOLD	I	The *address hold* request allows another bus master access to the Intel OverDrive processor's address bus for a cache invalidation cycle. The Intel OverDrive processor will stop driving its address bus in the clock following AHOLD going active. Only the address bus will be floated during address hold, the remainder of the bus will remain active. AHOLD is active HIGH and is provided with a small internal pulldown resistor. For proper operation AHOLD must meet setup and hold times t_{18} and t_{19}.
EADS#	I	This signal indicates that a *valid external address* has been driven onto the Intel OverDrive processor address pins. This address will be used to perform an internal cache invalidation cycle. EADS# is active LOW and is provided with an internal pullup resistor. EADS# must satisfy setup and hold times t_{12} and t_{13} for proper operation.
CACHE CONTROL		
KEN#	I	The *cache enable* pin is used to determine whether the current cycle is cacheable. When the Intel OverDrive processor generates a cycle that can be cached and KEN# is active, the cycle will become a cache line fill cycle. Returning KEN# active one clock before RDY# during the last read in the cache line fill will cause the line to be placed in the on-chip cache. KEN# is active LOW and is provided with a small internal pullup resistor. KEN# must satisfy setup and hold times t_{14} and t_{15} for proper operation.
FLUSH#	I	The *cache flush* input forces the Intel OverDrive processor to flush its entire internal cache. FLUSH# is active low and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition in any specific clock. FLUSH# being sampled low in the clock before the falling edge of RESET causes the Intel OverDrive processor to enter the tri-state test mode.
PAGE CACHEABILITY		
PWT PCD	O O	The *page write-through* and *page cache disable* pins reflect the state of the page attribute bits, PWT and PCD, in the page table entry or page directory entry. If paging is disabled or for cycles that are not paged, PWT and PCD reflect the state of the PWT and PCD bits in control register 3. PWT and PCD have the same timing as the cycle definition pins (M/IO#, D/C# and W/R#). PWT and PCD are active HIGH and are not driven during bus hold. PCD is masked by the cache disable bit (CD) in Control Register 0.
NUMERIC ERROR REPORTING		
FERR#	O	The *floating point error* pin is driven active when a floating point error occurs. FERR# is similar to the ERROR# pin on the Intel387™ math coprocessor. FERR# is included for compatibility with systems using DOS type floating point error reporting. FERR# will not go active if FP errors are masked in FPU register. FERR# is active LOW, and is not floated during bus hold.

1.4 PIN DESCRIPTIONS (Continued)

Symbol	Type	Name and Function
NUMERIC ERROR REPORTING (Continued)		
IGNNE#	I	When the *ignore numeric error* pin is asserted the Intel OverDrive processor will ignore a numeric error and continue executing non-control floating point instructions, but FERR# will still be activated by the Intel OverDrive processor. When IGNNE# is deasserted the Intel OverDrive processor will freeze on a non-control floating point instruction, if a previous floating point instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set. IGNNE# is active LOW and is provided with a small internal pullup resistor. IGNNE# is asynchronous but setup and hold times t_{20} and t_{21} must be met to insure recognition on any specific clock.
BUS SIZE CONTROL		
BS16# BS8#	I I	The *bus size 16* and *bus size 8* pins (bus sizing pins) cause the Intel OverDrive processor to run multiple bus cycles to complete a request from devices that cannot provide or accept 32 bits of data in a single cycle. The bus sizing pins are sampled every clock. The state of these pins in the clock before ready is used by the Intel OverDrive processor to determine the bus size. These signals are active LOW and are provided with internal pullup resistors. These inputs must satisfy setup and hold times t_{14} and t_{15} for proper operation.
ADDRESS MASK		
A20M#	I	When the *address bit 20 mask* pin is asserted, the Intel OverDrive processor masks physical address bit 20 (A20) before performing a lookup to the internal cache or driving a memory cycle on the bus. A20M# emulates the address wraparound at one Mbyte which occurs on the 8086. A20M# is active LOW and should be asserted only when the processor is in real mode. This pin is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For proper operation, A20M# should be sampled high at the falling edge of RESET.
i486 DX AND i486 SX CPU INTERFACE		
UP#(1,2)	O	The *upgrade present* pin is used to signal the Intel486 Microprocessor to float its outputs and get off the bus. It is active low and is never floated. UP# is driven low at power-up and remains active for the entire duration of the Upgrade Processor operation.
KEY PIN		
KEY(2)		The KEY pin is an electrically non-functional pin which is used to ensure correct Upgrade Processor orientation in a 169-pin socket.

NOTE:
1. The UP# pin was previously named the MP# pin in the i486 SX Microprocessor/i487 SX Math CoProcessor data book. The functionality is the same, only the name has changed.
2. The UP# input pin and KEY pin are not defined on the OverDrive processor for replacement of PGA Intel486 DX Microprocessor (ODPR).

Table 1.4.1. Output Pins

Name	Active Level	When Floated
BREQ	HIGH	
HLDA	HIGH	
BE0#–BE3#	LOW	Bus Hold
PWT, PCD	HIGH	Bus Hold
W/R#, D/C#, M/IO#	HIGH	Bus Hold
LOCK#	LOW	Bus Hold
PLOCK#	LOW	Bus Hold
ADS#	LOW	Bus Hold
BLAST#	LOW	Bus Hold
PCHK#	LOW	
FERR#	LOW	
SMIACT#	LOW	
UP#	LOW	
A2–A3	HIGH	Bus, Address Hold

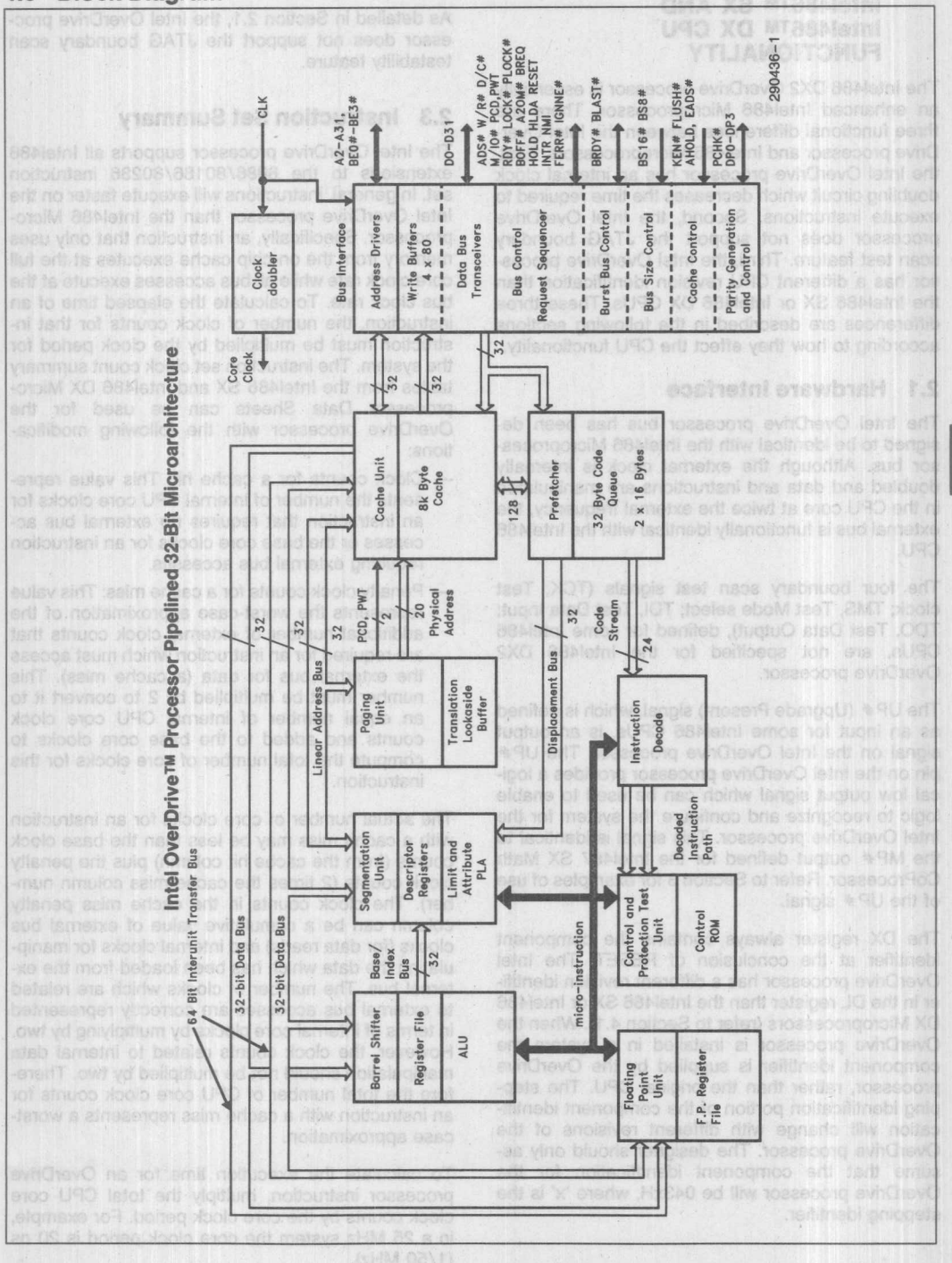
Table 1.4.2. Input Pins

Name	Active Level	Synchronous/ Asynchronous
CLK		
RESET	HIGH	Asynchronous
HOLD	HIGH	Synchronous
AHOLD	HIGH	Synchronous
EADS#	LOW	Synchronous
BOFF#	LOW	Synchronous
FLUSH#	LOW	Asynchronous
A20M#	LOW	Asynchronous
BS16#, BS8#	LOW	Synchronous
KEN#	LOW	Synchronous
RDY#	LOW	Synchronous
BRDY#	LOW	Synchronous
INTR	HIGH	Asynchronous
NMI	HIGH	Asynchronous
SRESET	HIGH	Asynchronous
SMI#	LOW	Asynchronous
STPCLK#	LOW	Asynchronous
IGNNE#	LOW	Asynchronous

Table 1.4.3. Input/Output Pins

Name	Active Level	When Floated
D0–D31	HIGH	Bus Hold
DP0–DP3	HIGH	Bus Hold
A4–A31	HIGH	Bus, Address Hold

1.5 Block Diagram



2.0 DIFFERENCES FROM Intel486™ SX AND Intel486™ DX CPU FUNCTIONALITY

The Intel486 DX2 OverDrive processor is essentially an enhanced Intel486 Microprocessor. There are three functional differences between the Intel OverDrive processor and Intel486 Microprocessors. First, the Intel OverDrive processor has an internal clock doubling circuit which decreases the time required to execute instructions. Second, the Intel OverDrive processor does not support the JTAG boundary scan test feature. Third, the Intel OverDrive processor has a different CPU revision identification than the Intel486 SX or Intel486 DX CPUs. These three differences are described in the following sections according to how they effect the CPU functionality.

2.1 Hardware Interface

The Intel OverDrive processor bus has been designed to be identical with the Intel486 Microprocessor bus. Although the external clock is internally doubled and data and instructions are manipulated in the CPU core at twice the external frequency, the external bus is functionally identical with the Intel486 CPU.

The four boundary scan test signals (TCK, Test clock; TMS, Test Mode select; TDI, Test Data Input; TDO, Test Data Output), defined for some Intel486 CPUs, are not specified for the Intel486 DX2 OverDrive processor.

The UP# (Upgrade Present) signal, which is defined as an input for some Intel486 CPUs, is an output signal on the Intel OverDrive processor. The UP# pin on the Intel OverDrive processor provides a logical low output signal which can be used to enable logic to recognize and configure the system for the Intel OverDrive processor. This signal is identical to the MP# output defined for the Intel487 SX Math CoProcessor. Refer to Section 3 for examples of use of the UP# signal.

The DX register always contains the component identifier at the conclusion of RESET. The Intel OverDrive processor has a different revision identifier in the DL register than the Intel486 SX or Intel486 DX Microprocessors (refer to Section 4.1). When the OverDrive processor is installed in a system the component identifier is supplied by the OverDrive processor, rather than the original CPU. The stepping identification portion of the component identification will change with different revisions of the OverDrive processor. The designer should only assume that the component identification for the OverDrive processor will be 043xH, where 'x' is the stepping identifier.

2.2 Testability

As detailed in Section 2.1, the Intel OverDrive processor does not support the JTAG boundary scan testability feature.

2.3 Instruction Set Summary

The Intel OverDrive processor supports all Intel486 extensions to the 8086/80186/80286 instruction set. In general, instructions will execute faster on the Intel OverDrive processor than the Intel486 Microprocessor. Specifically, an instruction that only uses memory from the on-chip cache executes at the full core clock rate while all bus accesses execute at the bus clock rate. To calculate the elapsed time of an instruction, the number of clock counts for that instruction must be multiplied by the clock period for the system. The instruction set clock count summary tables from the Intel486 SX and Intel486 DX Microprocessor Data Sheets can be used for the OverDrive processor with the following modifications:

— Clock counts for a cache hit: This value represents the number of internal CPU core clocks for an instruction that requires no external bus accesses or the base core clocks for an instruction requiring external bus accesses.

— Penalty clock counts for a cache miss: This value represents the worst-case approximation of the additional number of external clock counts that are required for an instruction which must access the external bus for data (a cache miss). This number must be multiplied by 2 to convert it to an equal number of internal CPU core clock counts and added to the base core clocks to compute the total number of core clocks for this instruction.

The actual number of core clocks for an instruction with a cache miss may be less than the base clock counts (from the cache hit column) plus the penalty clock counts (2 times the cache miss column number). The clock counts in the cache miss penalty column can be a cumulative value of external bus clocks (for data reads) and internal clocks for manipulating the data which has been loaded from the external bus. The number of clocks which are related to external bus accesses are correctly represented in terms of internal core clocks by multiplying by two. However, the clock counts related to internal data manipulation should not be multiplied by two. Therefore the total number of CPU core clock counts for an instruction with a cache miss represents a worst-case approximation.

To calculate the execution time for an OverDrive processor instruction, multiply the total CPU core clock counts by the core clock period. For example, in a 25 MHz system the core clock period is 20 ns (1/50 MHz).

PRELIMINARY

Additionally, the assumptions specified below should be understood in order to estimate instruction execution time.

A cache miss will force the OverDrive processor to run an external bus cycle. The Intel486 microprocessor 32-bit burst bus is defined as r−b−w.

Where:

r = The number of bus clocks in the first cycle of a burst read or the number of clocks per data cycle is a non-burst read.

b = The number of bus clocks for the second and subsequent cycles in a burst read.

w = The number of bus clocks for a write.

The fastest bus the OverDrive processor can support is 2−1−2 assuming 0 waits states. The clock counts in the cache miss penalty column assume a 2−1−2 bus. For slower busses add r−2 clocks to the cache miss penalty for the first dword accessed. Other factors also affect instruction clock counts.

Instruction Clock Count Assumptions

1. The external bus is available for reads or writes at all times. Else add bus clocks to reads until the bus is available
2. Accesses are aligned. Add three core clocks to each misaligned access.
3. Cache fills complete before subsequent accesses to the same line. If a read misses the cache during a cache fill due to a previous read or prefetch, the read must wait for the cache fill to complete. If a read or write accesses a cache line still being filled, it must wait for the fill to complete.
4. If an effective address is calculated, the base register is not the destination register of the preceding instruction. If the base register is the destination register of the preceding instruction add 1 to the core clock counts shown. Back-to-back PUSH and POP instructions are not affected by this rule.
5. An effective address calculation uses one base register and does not use an index register. However, if the effective address calculation uses an index register. 1 core clock may be added to the clock shown.
6. The target of a jump is in the cache. If not, add r clocks for accessing the destination instruction of a jump. If the destination instruction is not completely contained in the first dword read, add a maximum of 3b bus clocks. If the destination instruction is not completely contained in the first 16 byte burst, add a maximum of another r+3b bus clocks.
7. If no write buffer delay, w bus clocks are added only in the case in which all write buffers are full.
8. Displacement and immediate not used together. If displacement and immediate used together, 1 core clock may be added to the core clock count shown.
9. No invalidate cycles. Add a delay of 1 bus clock for each invalidate cycle if the invalidate cycle contends for the internal cache/external bus when the OverDrive processor needs to use it.
10. Page translation hits in TLB. A TLB miss will add 13, 21 or 28 bus clocks + 1 possible core clock to the instruction depending on whether the Accessed and/or Dirty bit in neither, one or both of the page entries needs to be set in memory. This assumes that neither page entry is in the data cache and a page fault does not occur on the address translation.
11. No exceptions are detected during instruction execution. Refer to interrupt core Clock Counts Table for extra clocks if an interrupt is detected.
12. Instructions that read multiple consecutive data items (i.e., task switch, POPA, etc.) and miss the cache are assumed to start the first access on a 16-byte boundary. If not, an extra cache line fill may be necessary which may add up to (r+3b) bus clocks to the cache miss penalty.

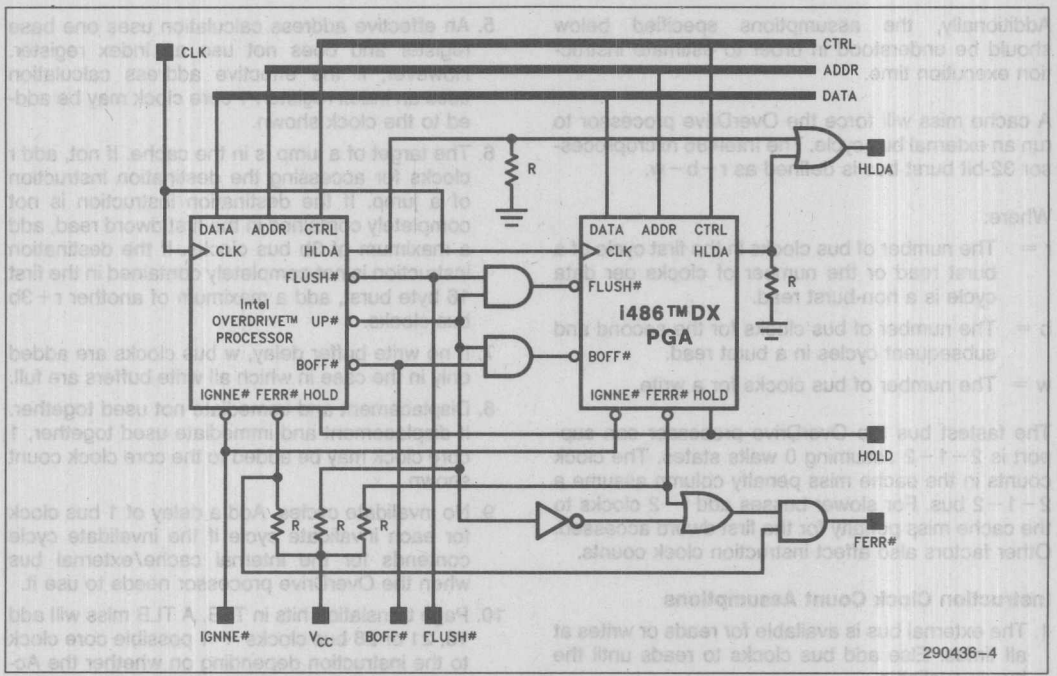


Figure 3.1. Intel OverDrive™ Socket Circuit Diagram for PGA Intel486™ DX CPU Based Systems

3.0 Intel OverDrive™ PROCESSOR CIRCUIT DESIGN

Figures 3.1, 3.2 and 3.3 show the circuits which interface the original CPU with the Intel OverDrive processor socket. These circuits allow Intel486 DX and Intel486 SX CPU based systems to be upgraded with the Intel OverDrive processor.

The circuits shown in Figures 3.1 and 3.2 may only be used for Intel486 CPUs that do not have the UP# input pin. The circuit shown in Figure 3.3 should be used for all Intel486 CPUs that have the UP# input pin.

3.1 Upgrade Circuit for PGA Intel486™ DX Based Systems

The Intel OverDrive processor socket Circuit for PGA Intel486 DX CPU based systems allows the Intel486 DX CPU complete control of the system when the Intel OverDrive processor socket is unpopulated. The HLDA signal from the Intel OverDrive processor socket should be tied low through a resistor while the UP# and FERR# signals from the Intel OverDrive processor socket should be tied high through a resistor to insure that the Intel486 DX CPU functions correctly when an Intel OverDrive processor socket component is not installed.

When the Intel OverDrive rocessor is installed, the Upgrade Present output, UP# pin, causes the FLUSH# and BOFF# signals to be driven active to the Intel486 DX CPU. When the Intel486 DX CPU samples FLUSH# active during reset, the Intel486 DX CPU enters tri-state output test mode after reset, which causes the Intel486 DX CPU to float all of its output signals. To float most of the Intel486 DX CPU's output pins before the end of reset, BOFF# is also driven active to the Intel486 DX CPU. BOFF# immediately causes all output signals to float except PCHK#, BREQ, HLDA and FERR#.

In addition to floating the PGA Intel486 DX CPU's outputs, the Intel486 DX CPU's HLDA and FERR# signals must be gated to prevent potential bus contention with the Intel OverDrive processor's HLDA and FERR# signals during reset. During reset the Intel486 DX CPU may not recognize HOLD active because BOFF# is driven active to the Intel486 DX CPU by the Intel OverDrive processor. If the Intel486 DX CPU does not recognize HOLD active, it will not drive HLDA active. However, the Intel OverDrive processor will recognize HOLD active and drive HLDA. By gating the HLDA signals from the Intel486 DX CPU and Intel OverDrive processor socket, bus contention is avoided if HOLD is driven active during reset. Because the state of FERR# is undefined during reset, bus contention is also avoided by gating FERR#.

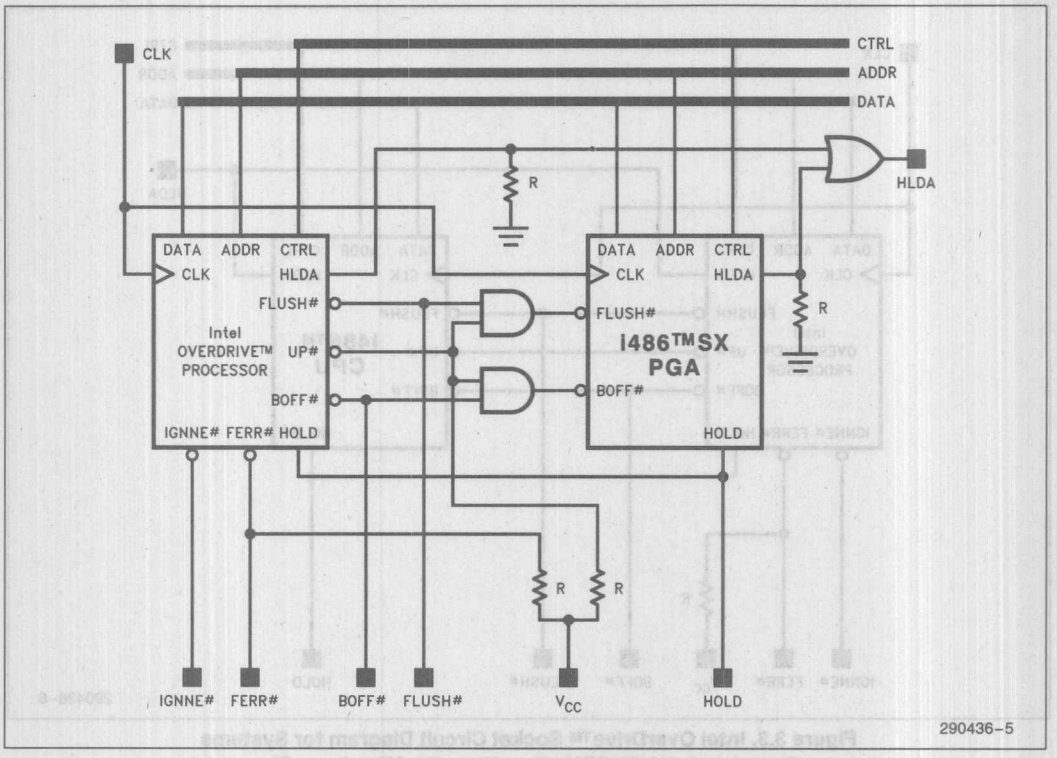


Figure 3.2. Intel OverDrive Socket Circuit Diagram for PGA Intel486 SX CPU Based Systems

3.2 Upgrade Circuit for PGA Intel486 SX CPU Based Systems

The Intel OverDrive processor socket Circuit for PGA Intel486 SX CPU based systems is the same as the Intel OverDrive processor socket Circuit for the PGA Intel486 DX CPU based system except that no gates are needed for the FERR# signal. The OverDrive processor socket Circuit allows the PGA Intel486 SX CPU complete control of the system when the Intel OverDrive processor socket is unpopulated. The HLDA signal from the Intel OverDrive processor socket should be tied low through a resistor while the UP# and FERR# signals from the Intel OverDrive processor socket should be tied high through a resistor to insure that the PGA Intel486 SX CPU functions correctly when an Intel OverDrive processor is not installed.

When the Intel OverDrive processor is installed, the Upgrade Present output, UP# pin, causes the FLUSH# and BOFF# signals to be driven active to the PGA Intel486 SX CPU. When the PGA Intel486 SX CPU samples FLUSH# active during reset, the PGA Intel486 SX CPU enters tri-state output test mode after reset which causes the PGA Intel486 SX CPU to float all of its output signals. To float most of the PGA Intel486 SX CPU's output pins before the end of reset, BOFF# is also driven active to the PGA Intel486 SX CPU. BOFF# immediately causes all output signals to float except PCHK#, BREQ, and HLDA.

In addition to floating the PGA Intel486 SX CPU's outputs, the PGA Intel486 SX CPU's HLDA signal must be gated to prevent potential bus contention with the Intel OverDrive processor's HLDA signal during reset. During reset, the PGA Intel486 SX CPU may not recognize HOLD active because BOFF# is driven active to the PGA Intel486 SX CPU by the Intel OverDrive processor. If the PGA Intel486 SX CPU does not recognize HOLD active, it will not drive HLDA active. However, the Intel OverDrive processor will recognize HOLD active and drive HLDA. By gating the HLDA signals from the PGA Intel486 SX CPU and Intel OverDrive processor, bus contention is avoided if HOLD is driven active during reset.

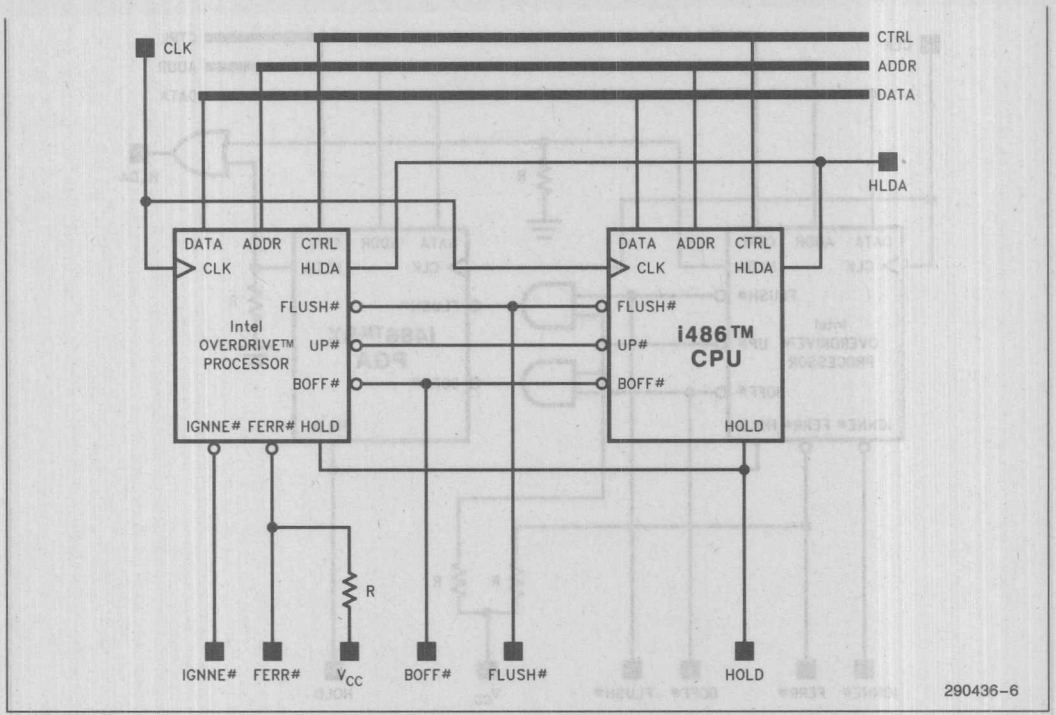


Figure 3.3. Intel OverDrive™ Socket Circuit Diagram for Systems Based on Intel486 CPUs That Have the UP# Input Pin

3.3 Upgrade Circuit for Intel486 CPU Based Systems with UP#

The Intel OverDrive processor socket circuit for UP# Intel486 CPU based systems requires no additional gates. The Upgrade Present input, UP# pin, allows the Intel486 CPU to directly recognize when the Intel OverDrive processor socket is populated. When the UP# pin is driven active to the Intel486 CPU, the Intel486 CPU tri-states all of its output pins and enters power-down mode.

4.0 BIOS AND SOFTWARE

The following should be considered when designing a system for upgrade with an Intel OverDrive processor.

4.1 Intel OverDrive Processor Detection

The component identifier and stepping/revision identifier for the Intel OverDrive processor is readable in DH and DL registers respectively, immediately after RESET, where

DH = 04h

DL = 30h–3Fh.

As it is difficult to differentiate between the Intel486 DX CPU and the Intel OverDrive processor in software, it is recommended that the BIOS save the contents of the DX register, immediately after RESET, so that this information can be used later, if required, to identify an Intel OverDrive processor in the system.

NOTE:
Initialization routines for Intel486 SX CPU systems should check for the presence of a floating point unit and set the CR0 register accordingly. (Refer to the Intel486 SX Microprocessor Data Book for specific details.)

4.2 Timing Dependent Loops

The Intel OverDrive processor executes instructions at twice the frequency of the input clock. Thus, software (or instruction based) timing loops will execute faster on the Intel OverDrive processor than on the Intel486 DX or Intel486 SX CPU (at the same input clock frequency). Instructions such as NOP, LOOP, and JMP $+2, have been used by BIOS to implement timing loops that are required, for example, to enforce recovery time between consecutive accesses for I/O devices. These instruction based timing loop implementations may require modification for systems intended to be upgradable with the Intel OverDrive processor.

In order to avoid any incompatibilities, it is recommended that timing requirements be implemented in hardware rather than in software. This provides transparency and also does not require any change in BIOS or I/O device drivers in the future when moving to higher processor clock speeds. As an example, a timing routine may be implemented as follows: The software performs a dummy I/O instruction to an unused I/O port. The hardware for the bus controller logic recognizes this I/O instruction and delays the termination of the I/O cycle to the CPU by keeping RDY# or BRDY# deasserted for the appropriate amount of time.

5.0 ELECTRICAL DATA

The following sections describe recommended electrical connections for the Intel OverDrive processor, and its electrical specifications.

5.1 Power and Grounding

5.1.1 POWER CONNECTIONS

Power and ground connections must be made to all external V_{CC} and GND pins of the Intel OverDrive processor. On the circuit board, all V_{CC} pins must be connected on a V_{CC} plane. All V_{SS} pins must be likewise connected on a GND plane.

5.1.2 POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitance should be placed near the Intel OverDrive processor. The Intel OverDrive processor driving its 32-bit parallel address and data busses at high frequencies can cause transient power surges, particularly when driving large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the Intel OverDrive processor and decoupling capacitors as much as possible. Capacitors specifically for PGA packages are also commercially available.

5.1.3 OTHER CONNECTION RECOMMENDATIONS

N.C. pins should always remain unconnected.

For reliable operation, always connect unused inputs to an appropriate signal level. Active LOW inputs should be connected to V_{CC} through a pullup resistor. Pullups in the range of 20 KΩ are recommended. Active HIGH inputs should be connected to GND.

5.2 Maximum Ratings

Table 5.1 is a stress rating only, and functional operation at the maximums is not guaranteed. Function operating conditions are given in Section 5.3 D.C. Specifications and Section 5.4 A.C. Specifications.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the Intel OverDrive processor contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

Table 5.1. Absolute Maximum Ratings

Case Temperature under Bias	−65°C to +110°C
Storage Temperature	−65°C to +150°C
Voltage on Any Pin with Respect to Ground	−0.5 to V_{CC} + 0.5V
Supply Voltage with Respect to V_{SS}	−0.5V to +6.5V

5.3 D.C. Specifications

Functional Operating Range: V_{CC} = 5V ±5%; (Note 1)

Table 5-2. DC Parametric Values

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 2)
V_{OH}	Output High Voltage	2.4		V	(Note 3)
I_{CC}	Power Supply Current CLK = 33 MHz CLK = 25 MHz CLK = 20 MHz CLK = 16 MHz		 1200 950 775 625	 mA	 (Note 4)
I_{LI}	Input Leakage Current		±15	μA	(Note 5)
I_{IH}	Input Leakage Current		200	μA	(Note 6)
I_{IL}	Input Leakage Current		−400	μA	(Note 7)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		13	pF	F_C = 1 MHz (Note 8)
C_O	I/O or Output Capacitance		17	pF	F_C = 1 MHz (Note 8)
C_{CLK}	CLK Capacitance		15	pF	F_C = 1 MHz (Note 8)

PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
1. The function operating temperature range is:
   OverDrive processor—20 MHz, T_{case} = 0°C to +95°C
   OverDrive processor—25 MHz, T_{sink} = 0°C to +85°C
   OverDrive processor—33 MHz, T_{sink} = 0°C to +85°C
2. This parameter is measured at:
   Address, Data, BEn 4.0 mA
   Definition, Control 5.0 mA
3. This parameter is measured at:
   Address, Data, BEn −1.0 mA
   Definition, Control −0.9 mA
4. Typical supply current:
   525 mA @ CLK = 16 MHz
   625 mA @ CLK = 20 MHz
   775 mA @ CLK = 25 MHz
   975 mA @ CLK = 33 MHz
5. This parameter is for inputs without internal pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
6. This parameter is for inputs with internal pulldowns and V_{IH} = 2.4V.
7. This parameter is for inputs with internal pullups and V_{IL} = 0.45V.
8. Not 100% tested.

5.4 A.C. Specifications

The A.C. specifications, given in Tables 5.3.1, 5.3.2, 5.3.3, and 5.3.4, consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the rising edge of the CLK signal.

A.C. specifications measurement is defined by Figures 5.1–5.6. All timings are referenced to 1.5V unless otherwise specified. Inputs must be driven to the voltage levels indicated by Figure 5.3 when A.C. specifications are measured. Intel OverDrive processor output delays are specified with minimum and maximum limits, measured as shown. The minimum Intel OverDrive processor delay times are hold times provided to external circuitry. Intel OverDrive processor input setup and hold times are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct Intel OverDrive processor operation.

V_{CC} = 5V ±5%; T_{sink} = 0°C to +85°C; C_l = 50 pF unless otherwise specified (Note 2)

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	33	MHz		1X Clock Driven to OverDrive processor
t_1	CLK Period	30	125	ns	5.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	11		ns	5.1	at 2V
t_3	CLK Low Time	11		ns	5.1	at 0.8V
t_4	CLK Fall Time		3	ns	5.1	2V to 0.8V
t_5	CLK Rise Time		3	ns	5.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, SMIACT#, FERR#, BREQ, HLDA Valid Delay	3	14	ns	5.5	(Note 3)
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		20	ns	5.6	(Note 1)
t_8	PCHK# Valid Delay	3	22	ns	5.4	(Note 3)
t_{8a}	BLAST#, PLOCK# Valid Delay	3	20	ns	5.5	(Note 3)
t_9	BLAST#, PLOCK# Float Delay		20	ns	5.6	(Note 1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	18	ns	5.5	(Note 3)
t_{11}	D0–D31, DP0–3 Write Data Float Delay		20	ns	5.6	(Note 1)
t_{12}	EADS# Setup Time	5		ns	5.2	
t_{13}	EADS# Hold Time	3		ns	5.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	5		ns	5.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	5.2	
t_{16}	RDY#, BRDY# Setup Time	5		ns	5.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	5.3	
t_{18}	HOLD, AHOLD Setup Time	6		ns	5.2	
t_{18a}	BOFF# Setup Time	7		ns	5.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	5.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, IGNNE# Setup Time	5		ns	5.2	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, SMI#, STPCLK#, SRESET, IGNNE# Hold Time	3		ns	5.2	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	5		ns	5.2, 5.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	5.2, 5.3	

PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
1. Not 100% tested. Guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF. Charts 5.7.1–5.7.3 provides the charts that may be used to determine the delay due to derating, depending on the lumped capacitive loading, that must be added to these specification values.
3. The minimum Intel OverDrive processor output valid delays are hold times provided to external circuitry.
4. A reset pulse width of 15 CLK cycles is required for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.

Table 5.3.2. 25 MHz Intel OverDrive Processor A.C. Characteristics

V_{CC} = 5V ±5%; T_{sink} = 0°C to +85°C; C_l = 50 pF unless otherwise specified (Note 2)

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	25	MHz		1X Clock Driven to OverDrive Processor
t_1	CLK Period	40	125	ns	5.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	14		ns	5.1	at 2V
t_3	CLK Low Time	14		ns	5.1	at 0.8V
t_4	CLK Fall Time		4	ns	5.1	2V to 0.8V
t_5	CLK Rise Time		4	ns	5.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA, SMIACT#, Valid Delay	3	19*	ns	5.5	(Note 3)
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		28	ns	5.6	(Note 1)
t_8	PCHK# Valid Delay	3	24	ns	5.4	(Note 3)
t_{8a}	BLAST#, PLOCK# Valid Delay	3	24	ns	5.5	(Note 3)
t_9	BLAST#, PLOCK# Float Delay		28	ns	5.6	(Note 1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	20	ns	5.5	(Note 3)
t_{11}	D0–D31, DP0–3 Write Data Float Delay		28	ns	5.6	(Note 1)
t_{12}	EADS# Setup Time	8		ns	5.2	
t_{13}	EADS# Hold Time	3		ns	5.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	8		ns	5.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	5.2	
t_{16}	RDY#, BRDY# Setup Time	8		ns	5.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	5.3	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	8		ns	5.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	5.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, SMI#, STPCLK#, SRESET, INTR, IGNNE# Setup Time	8		ns	5.2	
t_{21}	RESET, FLUSH#, A20M#, NMI, SMI#, STPCLK#, SRESET, INTR, IGNNE# Hold Time	3		ns	5.2	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	5		ns	5.2, 5.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	5.2, 5.3	

* PRELIMINARY SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
1. Not 100% tested. Guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF. Charts 5.7.1–5.7.3 provides the charts that may be used to determine the delay due to derating, depending on the lumped capacitive loading, that must be added to these specification values.
3. The minimum Intel OverDrive processor output valid delays are hold times provided to external circuitry.
4. A reset pulse width of 15 CLK cycles is required for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.

Table 5.3.3. 20 MHz Intel OverDrive Processor A.C. Characteristics

V_{CC} = 5V ±5%; T_{case} = 0°C to +95°C; C_l = 50 pF unless otherwise specified (Note 2)

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	20	MHz		1X Clock Driven to OverDrive Processor
t_1	CLK Period	50	125	ns	5.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	16		ns	5.1	at 2V
t_3	CLK Low Time	16		ns	5.1	at 0.8V
t_4	CLK Fall Time		6	ns	5.1	2V to 0.8V
t_5	CLK Rise Time		6	ns	5.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA, SMIACT# Valid Delay	3	23	ns	5.5	(Note 3)
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		37	ns	5.6	(Note 1)
t_8	PCHK# Valid Delay	3	26	ns	5.4	(Note 3)
t_{8a}	BLAST#, PLOCK# Valid Delay	3	28	ns	5.5	(Note 3)
t_9	BLAST#, PLOCK# Float Delay		37	ns	5.6	(Note 1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	26	ns	5.5	(Note 3)
t_{11}	D0–D31, DP0–3 Write Data Float Delay		37	ns	5.6	(Note 1)
t_{12}	EADS# Setup Time	10		ns	5.2	
t_{13}	EADS# Hold Time	3		ns	5.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	10		ns	5.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	5.2	
t_{16}	RDY#, BRDY# Setup Time	10		ns	5.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	5.3	
t_{18}	HOLD, AHOLD, Setup Time	12		ns	5.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	5.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, SMI#, STPCLK#, SRESET, INTR, IGNNE# Setup Time	12		ns	5.2	(Note 4)
t_{21}	RESET, FLUSH#, A20M#, NMI, SMI#, STPCLK#, SRESET, INTR, IGNNE# Hold Time	3		ns	5.2	(Note 4)
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	6		ns	5.2, 5.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	5.2, 5.3	

PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
1. Not 100% tested. Guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF. Charts 5.7.1–5.7.3 provides the charts that may be used to determine the delay due to derating, depending on the lumped capacitive loading, that must be added to these specification values.
3. The minimum Intel OverDrive processor output valid delays are hold times provided to external circuitry.
4. A reset pulse width of 15 CLK cycles is required for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.

Table 5.3.4. 16 MHz Intel OverDrive Processor A.C. Characteristics

V_{CC} = 5V ±5%; T_{case} = 0°C to +95°C; C_l = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	16	MHz		1X Clock Driven to OverDrive Processor
t_1	CLK Period	62.5	125	ns	5.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	20		ns	5.1	at 2V
t_3	CLK Low Time	20		ns	5.1	at 0.8V
t_4	CLK Fall Time		8	ns	5.1	2V to 0.8V
t_5	CLK Rise Time		8	ns	5.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA, SMIACT# Valid Delay	3	26*	ns	5.5	(Note 3)
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		42	ns	5.6	(Note 1)
t_8	PCHK# Valid Delay	3	35	ns	5.4	(Note 3)
t_{8a}	BLAST#, PLOCK# Valid Delay	3	35	ns	5.5	(Note 3)
t_9	BLAST#, PLOCK# Float Delay		42	ns	5.6	(Note 1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	30	ns	5.5	(Note 3)
t_{11}	D0–D31, DP0–3 Write Data Float Delay		42	ns	5.6	(Note 1)
t_{12}	EADS# Setup Time	12		ns	5.2	
t_{13}	EADS# Hold Time	4		ns	5.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	12		ns	5.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	4		ns	5.2	
t_{16}	RDY#, BRDY# Setup Time	12		ns	5.3	
t_{17}	RDY#, BRDY# Hold Time	4		ns	5.3	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	12		ns	5.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	4		ns	5.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, SMI#, STPCLK#, SRESET, INTR, IGNNE# Setup Time	14		ns	5.2	(Note 4)
t_{21}	RESET, FLUSH#, A20M#, NMI, SMI#, STPCLK#, SRESET, INTR, IGNNE# Hold Time	4		ns	5.2	(Note 4)
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	10		ns	5.2, 5.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	4		ns	5.2, 5.3	

PRELIMINARY
SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
1. Not 100% tested. Guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF. Charts 5.7.1–5.7.3 provides the charts that may be used to determine the delay due to derating, depending on the lumped capacitive loading, that must be added to these specification values.
3. The minimum Intel OverDrive processor output valid delays are hold times provided to external circuitry.
4. A reset pulse width of 15 CLK cycles is required for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.

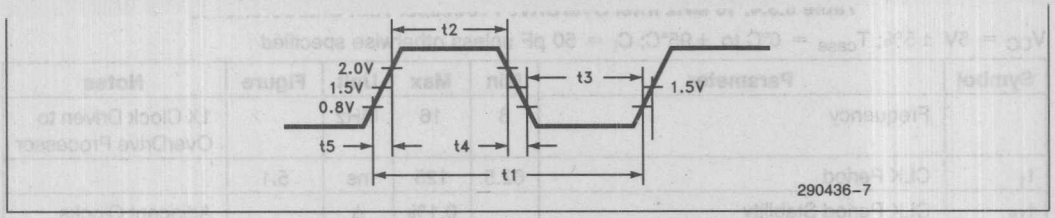


Figure 5.1. CLK Waveforms

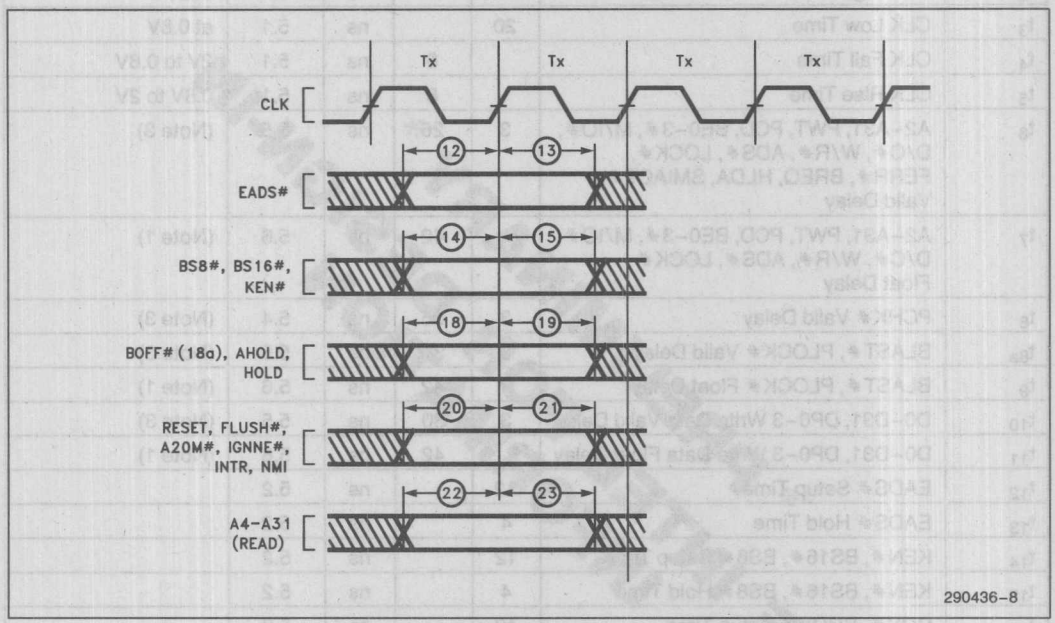


Figure 5.2. Input Setup and Hold Timing

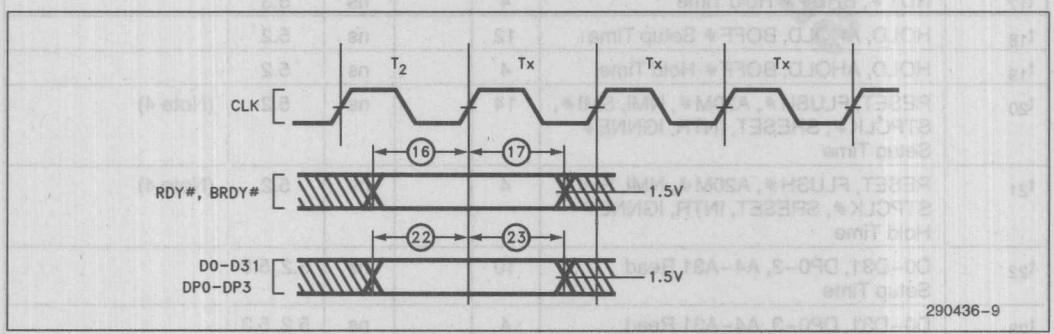


Figure 5.3. Input Setup and Hold Timing

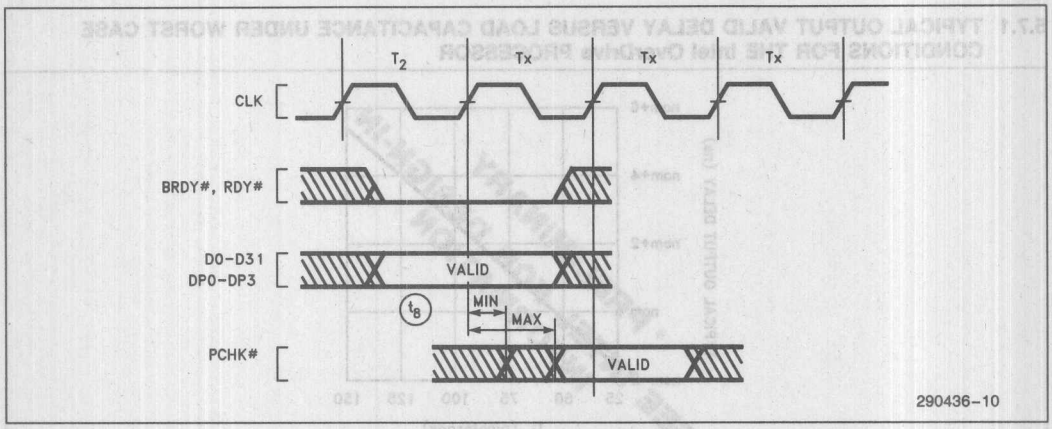


Figure 5.4. PCHK# Valid Delay Timing

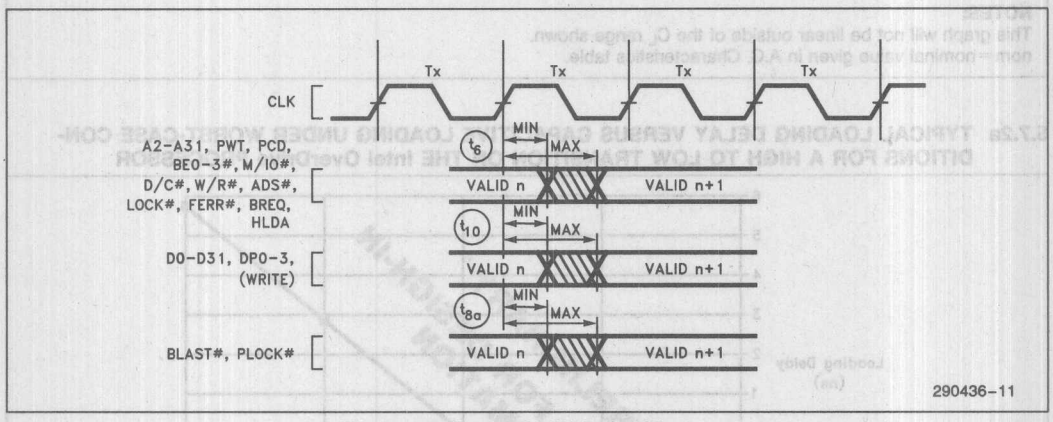


Figure 5.5. Output Valid Delay Timing

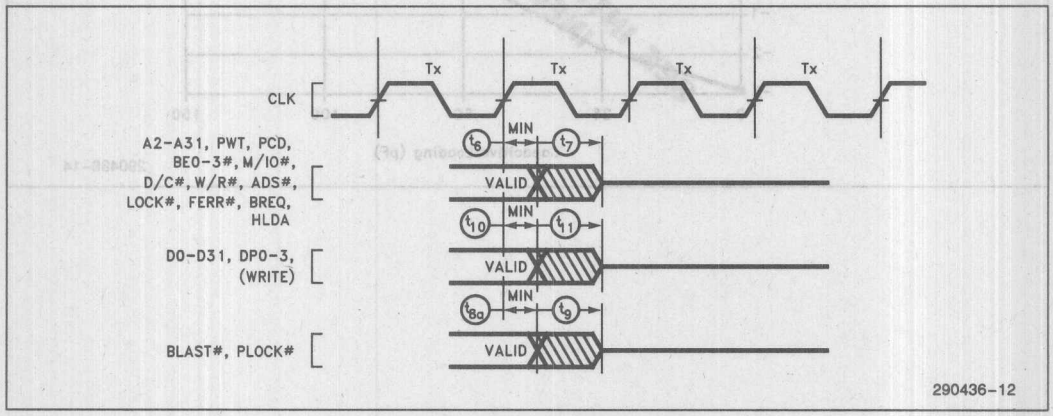
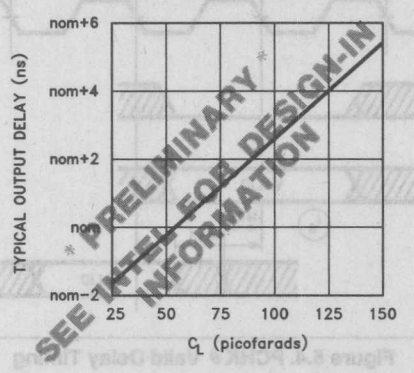


Figure 5.6. Maximum Float Delay Timing

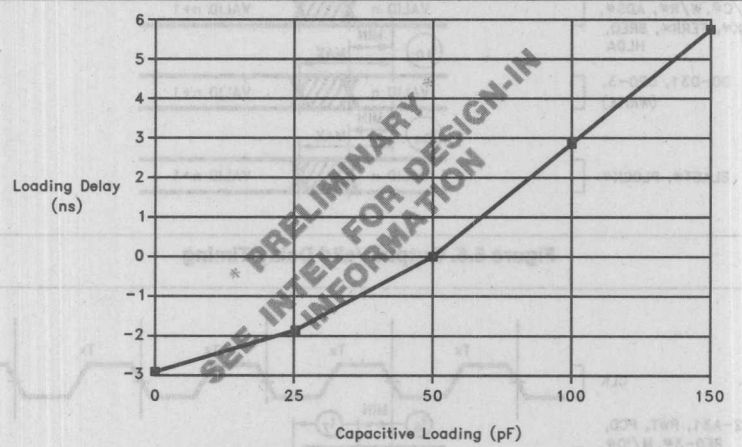
5.7.1 TYPICAL OUTPUT VALID DELAY VERSUS LOAD CAPACITANCE UNDER WORST CASE CONDITIONS FOR THE Intel OverDrive PROCESSOR



290436-13

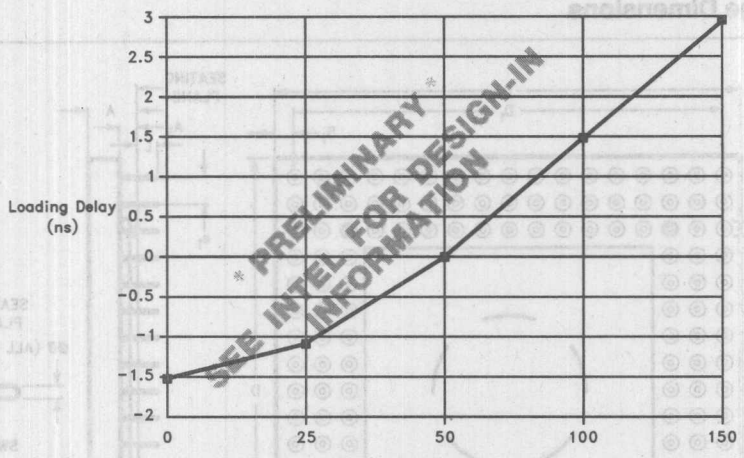
NOTES:
This graph will not be linear outside of the C_L range shown.
nom = nominal value given in A.C. Characteristics table.

5.7.2a TYPICAL LOADING DELAY VERSUS CAPACITIVE LOADING UNDER WORST-CASE CONDITIONS FOR A HIGH TO LOW TRANSITION ON THE Intel OverDrive PROCESSOR



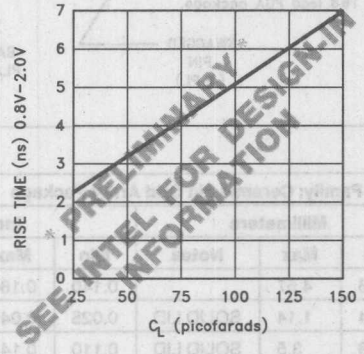
290436-14

5.7.2b TYPICAL LOADING DELAY VERSUS CAPACITIVE LOADING UNDER WORST-CASE CONDITIONS FOR A LOW TO HIGH TRANSITION ON THE Intel OverDrive PROCESSOR



290436-15

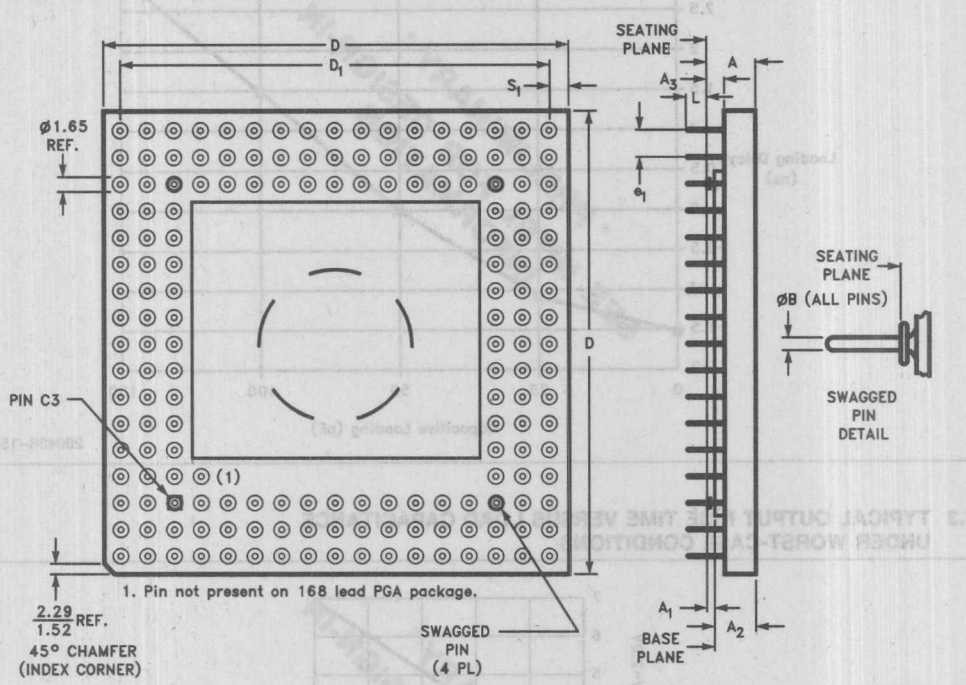
5.7.3 TYPICAL OUTPUT RISE TIME VERSUS LOAD CAPACITANCE UNDER WORST-CASE CONDITIONS



290436-16

NOTE:
This graph will not be linear outside of the C_L range shown.

6.1 Package Dimensions



Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		0.140	0.180	
A_1	0.64	1.14	SOLID LID	0.025	0.045	SOLID LID
A_2	2.8	3.5	SOLID LID	0.110	0.140	SOLID LID
A_3	1.14	1.40		0.045	0.055	
B	0.43	0.51		0.017	0.020	
D	44.07	44.83		1.735	1.765	
D_1	40.51	40.77		1.595	1.605	
e_1	2.29	2.79		0.090	0.110	
L	2.54	3.30		0.100	0.130	
N	168, 169			168, 169		
S_1	1.52	2.54		0.060	0.100	
ISSUE	IWS REV X 7/15/88					

Figure 6.1. 168 and 169 Lead Ceramic PGA Package Dimensions

Table 6.1. Ceramic PGA Package Dimension Symbols

Letter or Symbol	Description of Dimensions
A	Distance from seating plane to highest point of body
A_1	Distance between seating plane and base plane (lid)
A_2	Distance from base plane to highest point of body
A_3	Distance from seating plane to bottom of body
B	Diameter of terminal lead pin
D	Largest overall package dimension of length
D_1	A body length dimension, outer lead center to outer lead center
e_1	Linear spacing between true lead position centerlines
L	Distance from seating plane to end of lead
S_1	Other body dimension, outer lead center to edge of body

NOTES:
1. Controlling dimension: millimeter.
2. Dimension "e_1" ("e") is non-cumulative.
3. Seating plane (standoff) is defined by P.C. board hole size: 0.0415–0.0430 inch.
4. Dimensions "B", "B_1" and "C" are nominal.
5. Details of Pin 1 identifier are optional.

3

6.2 Heat Sink Dimensions

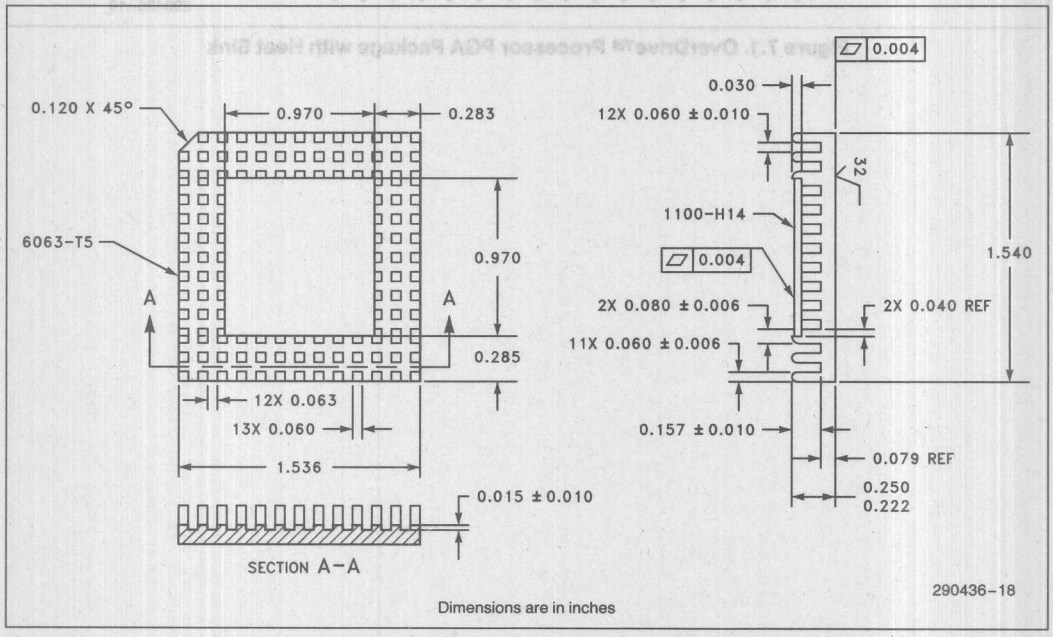


Figure 6.2. Intel OverDrive™ Processor Heat Sink Dimensions

7.0 THERMAL MANAGEMENT

The heat generated by the Intel OverDrive processor requires that heat dissipation be managed carefully. The 33 MHz and 25 MHz Intel OverDrive processors are supplied with a heat sink attached with adhesive to the package. 33 MHz and 25 MHz system designs must, therefore, provide space for the heat sink on the Intel OverDrive processor.

7.1 The Intel OverDrive™ Processor with Attached Heat Sink

The heat sink for the Intel OverDrive processor is adhesively attached to the standard PGA package. Figure 7.1 shows a drawing of the Intel OverDrive processor with the heat sink (see Table 7.1 for dimensions).

The maximum and minimum dimensions for the PGA package with heat sink are shown in Table 7.1.

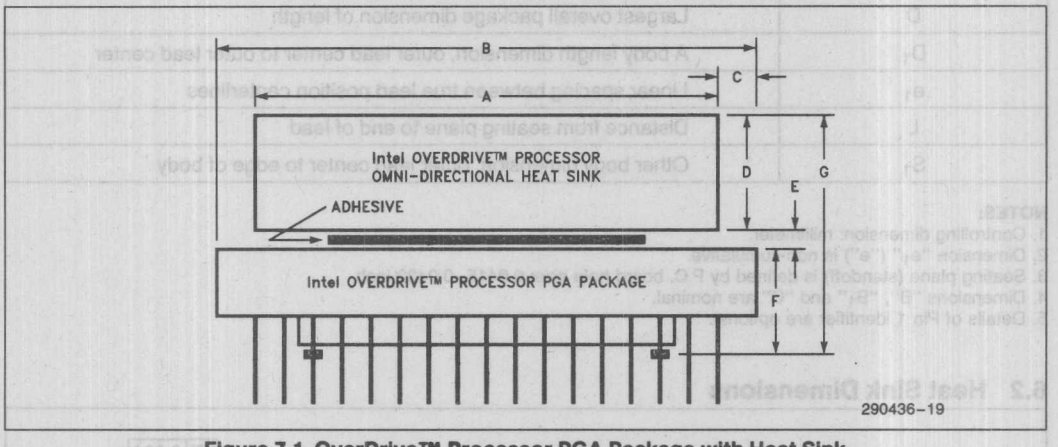


Figure 7.1. OverDrive™ Processor PGA Package with Heat Sink

Table 7.1. Intel OverDrive™ Processor PGA Package Dimensions with Heat Sink Attached

Dimension (Inches)	Min	Max
A. Heat Sink Width	1.520	1.550
B. PGA Package Width	1.735	1.765
C. Heat Sink Edge Gap	0.065	0.155
D. Heat Sink Height	0.212	0.260
E. Adhesive Thickness	0.008	0.012
F. Package Height from Stand-Offs	0.140	0.180
G. Total Height from Package Stand-Offs to Top of Heat Sink	0.360	0.452

The standard product markings and logo for the Intel OverDrive processor with the attached heat sink will be included on a 1 in^2 plate located on the top, center of the heat sink. The heat sink is omni-directional which allows the air to flow from any direction to achieve adequate cooling. The thermal resistance values for the Intel OverDrive processor with attached heat sink are shown in Table 7.2.

Table 7.2. Thermal Resistance for the Intel OverDrive™ Processor with Attached Heat Sink

θ_{JS} = 2.5°C/W	Airflow (Ft/min, LFM)				
	0	200	400	600	800
θ_{JA}(°C/W)	14.0*	10.0	7.5	6.2	5.7

NOTE:
*The thermal resistance from junction to ambient (θ_{JA}) in static air is actually a linear function of power dissipation. The value shown in the table (14.0°C/W) represents the worst case expected value which is derived from a power dissipation of 2.9W. The maximum expected power dissipation of 6W yields a θ_{JA} value of 13.1°C/W.

7.2 Intel OverDrive™ Processor without Heat Sink

The 20 MHz Intel OverDrive processor, for 16 MHz and 20 MHz i486 SX CPU systems, is supplied without a heat sink. Table 7.3 contains the thermal resistance values for the Intel OverDrive processor without heat sink.

Table 7.3. Thermal Resistance for the Intel OverDrive™ Processor without Heat Sink

θ_{JC} = 2.0°C/W	Airflow (Ft/min, LFM)				
	0	200	400	600	800
θ_{JA}(°C/W)	19.0*	16.0	12.5	11.0	10.0

NOTE:
*The thermal resistance from junction to ambient (θ_{JA}) in static air is actually a linear function of power dissipation. The value shown in the table (19.0°C/W) represents the worst case expected value which is derived from a power dissipation of 2.9W. The maximum expected power dissipation of 4W yields a θ_{JA} value of 18.5°C/W.

7.3 Thermal Equations

The methodology for calculating the heat dissipation for the 20 MHz Intel OverDrive processor (without a heat sink) and the 25 MHz or 33 MHz Intel OverDrive processor (with a heat sink) is identical except that the reference point for measuring the product temperature is different. The 20 MHz Intel OverDrive processor specifies T_{case} (the temperature at the outside center of the PGA package, opposite the pins—see Figure 7.2) and θ_{JC} (the thermal resistance from the silicon junction to the package case) but the 25 MHz and 33 MHz Intel OverDrive processor specifies T_{sink} (the temperature at the outside center base of the heat sink, not on the heat sink marking plate or cooling posts—see Figure 7.3) and θ_{JS} (the thermal resistance from the silicon junction to the heat sink base). The relationships between temperature, thermal resistance and power are shown in the following equations:

$$T_{case} = T_{ambient} + (P_{max} * \theta_{CA}) \text{ and}$$

$$T_{sink} = T_{ambient} + (P_{max} * \theta_{SA})$$

where,

$T_{ambient}$ = Ambient Temperature

P_{max} = Power ($I_{CC} * V_{CC}$),

$\theta_{CA} = \theta_{JA} - \theta_{JC}$,

$\theta_{SA} = \theta_{JA} - \theta_{JS}$.

3

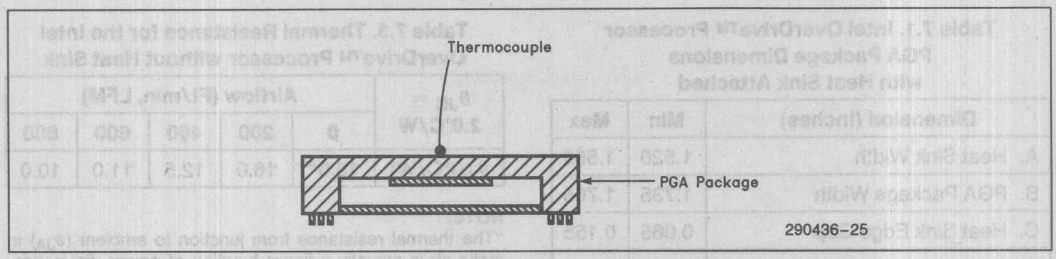


Figure 7.2. Case Temperature Measurement without Heat Sink (0.005" Dia. Thermocouple on the Center of Package Top Surface with a 90° Angle Adhesive Bond)

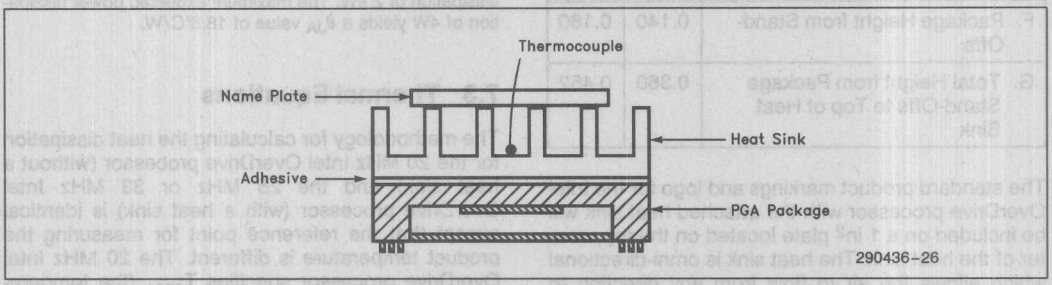


Figure 7.3. Heat Sink Measurement (0.005" Dia. Thermocouple on the Center of Heat Sink with a 90° Angle Adhesive Bond through a Hole Drilled through the Center of the Name Plate)

The maximum allowable ambient temperature as computed from these thermal equations is shown in Table 7.4.

Table 7.4. T_A (°C) for the OverDrive™ Processor

	f_{CLK} (MHz)	Airflow—Linear ft/min (m/sec)				
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
OverDrive Processor without Heat Sink(1)	16	42	51	62	67	70
	20	29	41	54	60	64
OverDrive Processor with Heat Sink(2)	25	30	49	61	67	70
	33	16	40	55	63	66

NOTES:
1. The 20 MHz OverDrive processor does not have a heat sink.
2. The 25 MHz and 33 MHz OverDrive processors have a heat sink attached.

APPENDIX A
"END USER EASY" UPGRADABILITY*

PC buyers value easy and safe upgrade installation. PC manufacturers can make the Intel OverDrive processor installation in the Intel OverDrive processor socket simple and foolproof for the end user and reseller by implementing the suggestions listed in Table A-1.

Table A-1. Socket and Layout Considerations

"End User Easy" Feature	Implementation
Visible OverDrive Processor Socket	The Intel OverDrive processor socket should be easily visible when the PC's cover is removed. Label the Intel OverDrive processor socket and the location of pin 1 by silk screening this information on the PC board.
Accessible OverDrive Processor Socket	Make the Intel OverDrive processor socket easily accessible to the end user (i.e., do not place the Intel OverDrive processor socket under a disk drive). If a Low Insertion Force (LIF) or screw machine socket is used, position the Intel OverDrive processor socket on the PC board such that there is ample clearance around the socket.
Foolproof Chip Orientation	Intel packages all Intel OverDrive processors in a 169-pin, PGA package. The 169th pin is called the "key" pin and insures that the Intel OverDrive processor fits into a 169-pin socket in only the correct orientation. Supplying a 169-pin socket as the Intel OverDrive processor socket eliminates the possibility of end users or resellers damaging the PC board or Intel OverDrive processor by powering up the system with the Intel OverDrive processor incorrectly oriented.
Zero Insertion Force Upgrade Socket	The high pin count of the Intel OverDrive processor makes the insertion force required for installation in a screw machine PGA socket excessive for end users or resellers. Even most Low Insertion Force (LIF) sockets often require more than 60 lbs. of insertion force. A Zero Insertion Force (ZIF) socket insures that the chip insertion force does not damage the PC board. If the ZIF socket has a handle, be sure to allow enough clearance for the socket handle. If a LIF or screw machine socket is used, additional PC board support is recommended.
"Plug and Play"	Jumper or switch changes should not be needed to electrically configure the system for the Intel OverDrive processor.
Thorough Documentation	Describe the Intel OverDrive processor's installation procedure in the PC's User's Manual.

*This section applies to the 169-lead PGA versions of the OverDrive processor only. The "End-User Easy" criteria have been incorporated into the OverDrive Ready program. OverDrive Ready is a trademark of Intel Corp.

APPENDIX B
ZIF and LIF SOCKET VENDORS

The following lists provide examples of sockets which can be used as the Intel OverDrive socket for Intel486 SX and Intel486 DX CPU based systems.

NOTE:
This is not a comprehensive list. Intel has not tested the sockets listed below and cannot guarantee that these sockets will meet every PC manufacturer's specific requirements.

Zero Insertion Force Upgrade Sockets and Vendors:

1. AMP Inc.
   P.O. Box 3608
   Harrisburg, PA 17105-3608
   Tel: (800) 522-6752
   Part Number: 55287-3
   Contact: Rick Simonic, New Product Manager
   (717) 561-6143

2. Augat Inc.
   425 John Dietsch Blvd.
   Attleboro Falls, MA 02763
   (508) 699-9800

3. Aries Electronics
   P.O. Box 130
   Frenchtown, NJ 08825
   Tel: (908) 996-6841
   Part Number: 169-PRS17012-10
   Contact: Frank Folmsbee, Marketing Manager
   (908) 996-6841

4. JAE
   599 N. Mathilda Ave., Suite 8
   Sunnyvale, CA 94086
   Tel: (408) 733-0493
   Part Number: PCPS-169-002
   Contact: Bob Gerleman, Western Sales Manager
   (408) 733-0493

5. Thomas and Betts
   200 Executive Center Drive
   P.O. Box 24901
   Greenville, SC 29616-2401
   Tel: (803) 676-2900
   Part Number: PGA169A17-S-1AC
   Contact: Scott Roland,
   Product Marketing Manager
   (803) 676-2910

6. Yamaichi Electronics
   1420 Koll Circle, Suite B
   San Jose, CA 95112
   Tel: (408) 452-0797
   Part Number: NP111-16911-G4
   Contact: Jim Bennett, Sales Manager
   (408) 452-0797

Low Insertion Force Sockets and Vendors:

1. AMP Inc.
   P.O. Box 3608
   Harrisburg, PA 17105-3608
   Tel: (800) 522-6752
   Part Number:
   (Premium Base Material) 55589-5
   (Standard Base Material) 916227-3

2. Thomas and Betts
   200 Executive Center Drive
   P.O. Box 24901
   Greenville, SC 29616-2401
   Tel: (803) 676-2900
   Part Number: LPG169A17-S-1AC

Peripheral Components

4

4 Peripheral Components

82091AA
ADVANCED INTEGRATED PERIPHERAL (AIP)

- **Single-Chip PC Compatible I/O Solution for Notebook and Desktop Platforms:**
  - 82078 Floppy Disk Controller Core
  - Two 16550 Compatible UARTs
  - One Multi-Function Parallel Port
  - IDE Interface
  - Integrated Back Power Protection
  - Integrated Game Port Chip Select
  - 5V or 3.3V Supply Operation with 5V Tolerant Drive Interface
  - Full Power Management Support
  - Supports Type F DMA Transfers for Faster I/O Performance
  - No Wait-State Host I/O Interface
  - Programmable Interrupt Interfaces
  - Single Crystal/Oscillator Clock (24 MHz)
  - Software Detectable Device ID
  - Comprehensive Powerup Configuration
- **The AIP is 100% Compatible with EISA, ISA and AT**
- **Host Interface Features**
  - 8-Bit Zero Wait-State ISA Bus Interface
  - DMA with Type F Transfers
  - Five Programmable ISA Interrupt Lines
  - Internal Address Decoder
- **Parallel Port Features**
  - All IEEE Standard 1284 Protocols Supported (Compatibility, Nibble, Byte, EPP, and ECP)
  - Peak Bi-Directional Transfer Rate of 2 MB/sec
  - 16 Byte FIFO for ECP
- **Floppy Disk Controller Features**
  - 100% Software Compatible with Industry Standard 82077SL and 82078
  - Integrated Analog Data Separator 250K, 300K, 500K, 1M
  - Programmable Powerdown Command
  - Auto Powerdown and Wakeup Modes
  - Integrated Tape Drive Support
  - Perpendicular Recording Support for 4 MB Drives
  - Programmable Write Pre-Compensation Delays
  - 256 Track Direct Address, Unlimited Track Support
  - 16 Byte FIFO
  - Supports 2 or 4 Drives
- **16550 Compatible UART Features**
  - Two Independent Serial Ports
  - Software Compatible with 8250 and 16450 UARTs
  - 16 Byte FIFO per Serial Port
  - Two UART Clock Sources, Supports MIDI Baud Rate
- **IDE Interface Features**
  - Generates Chip Selects for IDE Drives
  - Integrated Buffer Control Logic
  - Dual IDE Interface Support
- **Power Management Features**
  - Transparent to Operating Systems and Applications Programs
  - Independent Power Control for Each Integrated Device
- **100-Pin QFP Package**
  See Packaging Spec. 240800

The 82091AA Advanced Integrated Peripheral (AIP) is an integrated I/O solution containing a floppy disk controller, 2 serial ports, a multi-function parallel port, an IDE interface, and a game port on a single chip. The integration of these I/O devices results in a minimization of form factor, cost and power consumption. The floppy disk controller is the 82078 core with a data rate up to 2 Mbs. The serial ports are 16550 compatible. The parallel port supports all of the IEEE Standard 1284 protocols (ECP, EPP, Byte, Compatibility, and Nibble). The IDE interface supports 8-bit or 16-bit programmed I/O and 16-bit DMA. The Host Interface is an 8-bit ISA

October 1993
Order Number: 290486-001

interface optimized for type "F" DMA and no wait-state I/O accesses. Improved throughput and performance, the AIP contains six 16-byte FIFOstwo for each serial port, one for the parallel port, and one for the floppy disk controller. The AIP also includes power management and 3.3V capability for power sensitive applications such as notebooks. The AIP supports both motherboard and add-in card configurations.

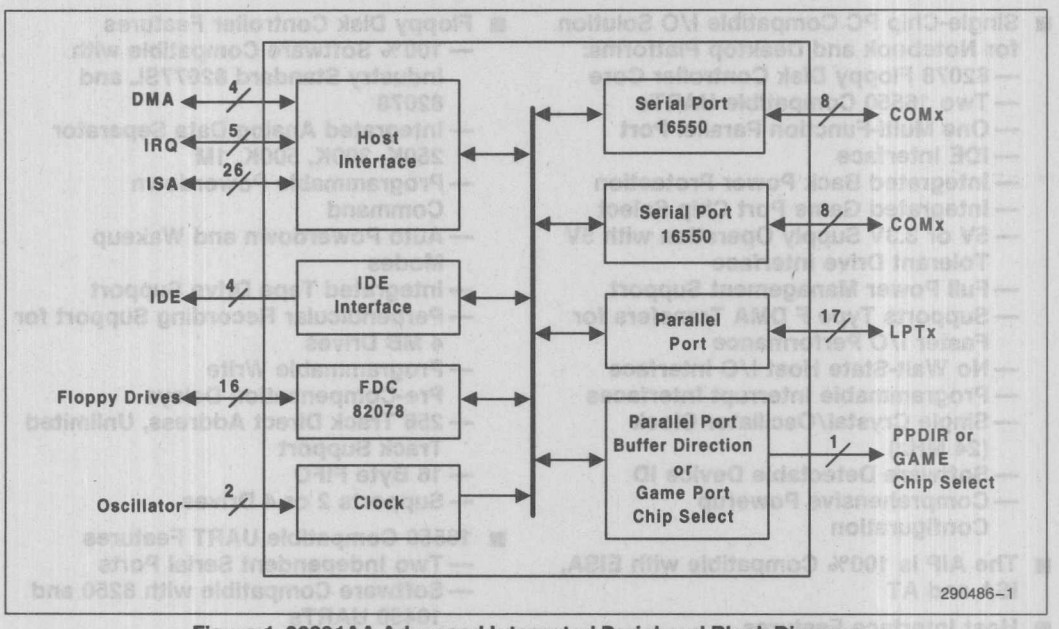


Figure 1. 82091AA Advanced Integrated Peripheral Block Diagram

82078 CHMOS SINGLE-CHIP FLOPPY DISK CONTROLLER

- **Small Footprint and Low Height Packages**
- **Supports Standard 5.0V as Well as Low Voltage 3.3V Platforms**
  - **— Selectable 3.3V and 5.0V Configuration**
  - **— 5.0V Tolerant Drive Interface**
- **Enhanced Power Management**
  - **— Application Software Transparency**
  - **— Programmable Powerdown Command**
  - **— Save and Restore Commands for 0V Powerdown**
  - **— Auto Powerdown and Wakeup Modes**
  - **— Two External Power Management Pins**
  - **— Consumes No Power While in Powerdown**
- **Programmable Internal Oscillator**
- **Floppy Drive Support Features**
  - **— Drive Specification Command**
  - **— Media ID Capability Provides Media Recognition**
  - **— Drive ID Capability Allows the User to Recognize the Type of Drive**
  - **— Selectable Boot Drive**
  - **— Standard IBM and ISO Format Features**
  - **— Format with Write Command for High Performance in Mass Floppy Duplication**
- **Integrated Host/Disk Interface Drivers**
- **Integrated Analog Data Separator**
  - **— 250 Kbits/sec**
  - **— 300 Kbits/ sec**
  - **— 500 Kbits/ sec**
  - **— 1 Mbits/sec**
  - **— 2 Mbits/sec**
- **Integrated Tape Drive Support**
  - **— Standard 1 Mbps/500 Kbps/ 250 Kbps Tape Drives**
  - **— New 2 Mbps Tape Drive Mode**
- **Perpendicular Recording Support for 4 MB Drives**
- **Fully Decoded Drive Select and Motor Signals**
- **Programmable Write Precompensation Delays**
- **Addresses 256 Tracks Directly, Supports Unlimited Tracks**
- **16 Byte FIFO**
- **Single-Chip Floppy Disk Controller Solution for Portables and Desktops**
  - **— 100% PC-AT* Compatible**
  - **— 100% PS/2* Compatible**
  - **— 100% PS/2 Model 30 Compatible**
  - **— Fully Compatible with Intel's 386SL Microprocessor SuperSet**
  - **— Integrated Drive and Data Bus Buffers**
- **Available in 64 Pin QFP and 44 Pin QFP Package**
  (See Package Specification Order Number 240800, Package Type S)

The 82078 Product Family brings a set of enhanced floppy disk controllers. These include several features that allow for easy implementation in both the portable and desktop market. The current family includes a 64 pin and a 44 pin part in the smaller form factor QFP package. The 3.3V version of the 64 pin part provides an ideal solution for the rapidly emerging 3.3V platforms. It also allows for a 5.0V tolerant floppy drive interface that lets the users retain their normal 5.0V drives. Another version of the 64 pin part provides support for 2 Mbps data rate tape drives.

*Other brands and names are the property of their respective owners.

Table 1-0. 64 Pin Part Versions

	3.3V	5.0V	2 Mbps Data Rate
82078SL	X	X	
82078-1		X	X

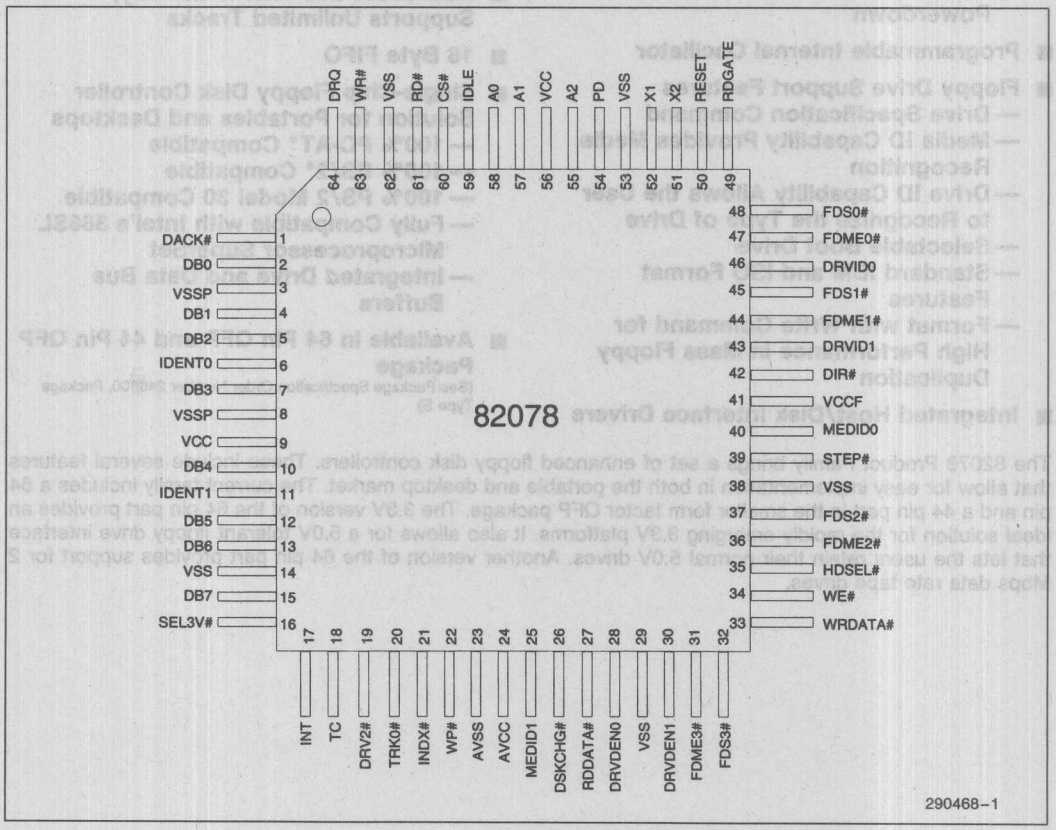
The 44 pin is targeted for platforms that are operated at 3.3V or 5.0V and do not require more than two drive support. The 82078-5 is designed for price sensitive 5.0V designs which do not include 4 MB drive support.

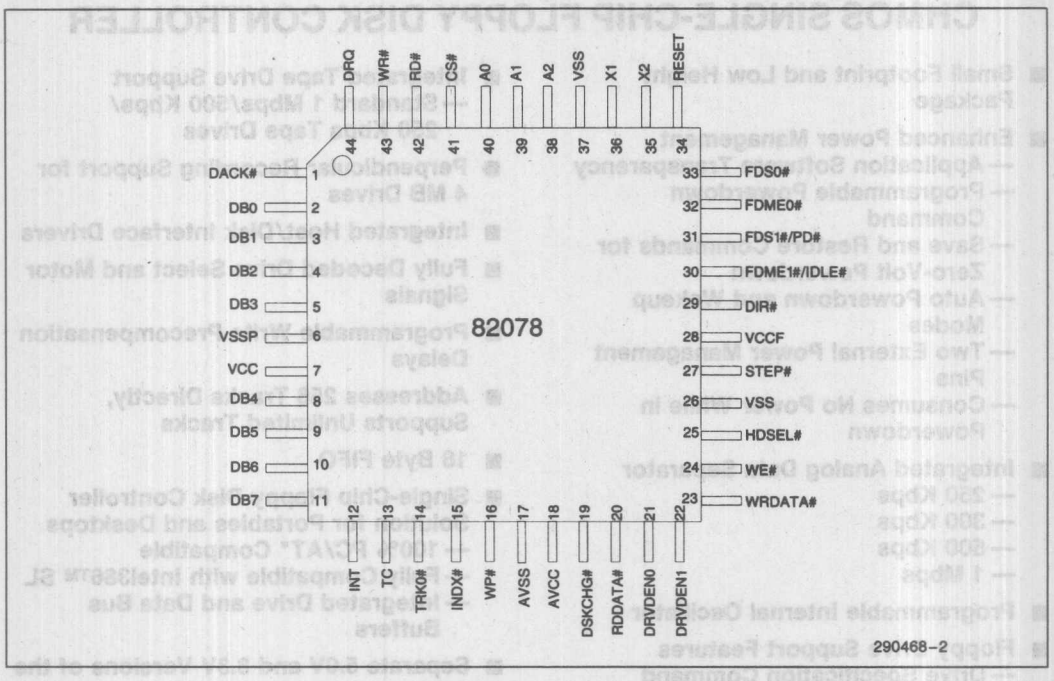
Table 2-0. 44 Pin Part Versions

	3.3V	5.0V	1 Mbps Data Rate
82078		X	X
82078-5		X	
82078-3	X		X

Both parts can be operated at 1 Mbps/500 Kbps/300 Kbps/250 Kbps. Additionally, one version of the 64 pin part provides 2 Mbps data rate operation specific for the new tape drives.

The 82078 is fabricated with Intel's advanced CHMOS III technology.




82078
DRQ
WR#
RD#
CS#
A0
A1
A2
VSS
X1
X2
RESET
44
43
42
41
40
39
38
37
36
35
34
DACK# 1
DB0 2
DB1 3
DB2 4
DB3 5
VSSP 6
VCC 7
DB4 8
DB5 9
DB6 10
DB7 11
33 FDS0#
32 FDME0#
31 FDS1#/PD#
30 FDME1#/IDLE#
29 DIR#
28 VCCF
27 STEP#
26 VSS
25 HDSEL#
24 WE#
23 WRDATA#
12
13
14
15
16
17
18
19
20
21
22
INT
TC
TRK0#
INDX#
WP#
AVSS
AVCC
DSKCHG#
RDDATA#
DRVDEN0
DRVDEN1
290468-2

82078 44 PIN
CHMOS SINGLE-CHIP FLOPPY DISK CONTROLLER

- **Small Footprint and Low Height Package**
- **Enhanced Power Management**
  - **— Application Software Transparency**
  - **— Programmable Powerdown Command**
  - **— Save and Restore Commands for Zero-Volt Powerdown**
  - **— Auto Powerdown and Wakeup Modes**
  - **— Two External Power Management Pins**
  - **— Consumes No Power While in Powerdown**
- **Integrated Analog Data Separator**
  - **— 250 Kbps**
  - **— 300 Kbps**
  - **— 500 Kbps**
  - **— 1 Mbps**
- **Programmable Internal Oscillator**
- **Floppy Drive Support Features**
  - **— Drive Specification Command**
  - **— Selectable Boot Drive**
  - **— Standard IBM and ISO Format Features**
  - **— Format with Write Command for High Performance in Mass Floppy Duplication**
- **Integrated Tape Drive Support**
  - **— Standard 1 Mbps/500 Kbps/ 250 Kbps Tape Drives**
- **Perpendicular Recording Support for 4 MB Drives**
- **Integrated Host/Disk Interface Drivers**
- **Fully Decoded Drive Select and Motor Signals**
- **Programmable Write Precompensation Delays**
- **Addresses 256 Tracks Directly, Supports Unlimited Tracks**
- **16 Byte FIFO**
- **Single-Chip Floppy Disk Controller Solution for Portables and Desktops**
  - **— 100% PC/AT* Compatible**
  - **— Fully Compatible with Intel386™ SL**
  - **— Integrated Drive and Data Bus Buffers**
- **Separate 5.0V and 3.3V Versions of the 44 Pin part are Available**
- **Available in a 44 Pin QFP Package**

The 82078, a 24 MHz crystal, a resistor package, and a device chip select implements a complete solution. All programmable options default to 82078 compatible values. The dual PLL data separator has better performance than most board level/discrete PLL implementations. The FIFO allows better system performance in multi-master (e.g., Microchannel, EISA).

The 82078 maintains complete software compatibility with the 82077SL/82077AA/8272A floppy disk controllers. It contains programmable power management features while integrating all of the logic required for floppy disk control. The power management features are transparent to any application software.

The 82078 is fabricated with Intel's advanced CHMOS III technology and is also available in a 64-lead QFP package.

---

*Other brands and names are the property of their respective owners.

Refer to the 1994 Peripheral Components Handbook for the complete data sheet on this device.

October 1993
Order Number: 290474-003

intel®

82078 64 PIN
CHMOS SINGLE-CHIP FLOPPY DISK CONTROLLER

- Small Footprint and Low Height Packages
- Supports Standard 5.0V as well as Low Voltage 3.3V Platforms
  - — Selectable 3.3V and 5.0V Configuration
  - — 5.0V Tolerant Drive Interface
- Enhanced Power Management
  - — Application Software Transparency
  - — Programmable Powerdown Command
  - — Save and Restore Commands for Zero-Volt Powerdown
  - — Auto Powerdown and Wakeup Modes
  - — Two External Power Management Pins
  - — Consumes no Power when in Powerdown
- Integrated Analog Data Separator
  - — 250 Kbps
  - — 300 Kbps
  - — 500 Kbps
  - — 1 Mbps
  - — 2 Mbps
- Programmable Internal Oscillator
- Floppy Drive Support Features
  - — Drive Specification Command
  - — Media ID Capability Provides Media Recognition
  - — Drive ID Capability Allows the User to Recognize the Type of Drive
  - — Selectable Boot Drive
  - — Standard IBM and ISO Format Features
  - — Format with Write Command for High Performance in Mass Floppy Duplication
- Integrated Tape Drive Support
  - — Standard 1 Mbps/500 Kbps/ 250 Kbps Tape Drives
  - — New 2 Mbps Tape Drive Mode
- Perpendicular Recording Support for 4 MB Drives
- Integrated Host/Disk Interface Drivers
- Fully Decoded Drive Select and Motor Signals
- Programmable Write Precompensation Delays
- Addresses 256 Tracks Directly, Supports Unlimited Tracks
- 16 Byte FIFO
- Single-Chip Floppy Disk Controller Solution for Portables and Desktops
  - — 100% PC AT* Compatible
  - — 100% PS/2* Compatible
  - — 100% PS/2 Model 30 Compatible
  - — Fully Compatible with Intel386™ SL Microprocessor SuperSet
- Integrated Drive and Data Bus Buffers
- Available in 64 Pin QFP Package

4

The 82078, a 24 MHz crystal, a resistor package, and a device chip select implements a complete solution. All programmable options default to 82078 compatible values. The dual PLL data separator has better performance than most board level/discrete PLL implementations. The FIFO allows better system performance in multi-master (e.g., Microchannel, EISA).

The 82078 maintains complete software compatibility with the 82077SL/82077AA/8272A floppy disk controllers. It contains programmable power management features while integrating all of the logic required for floppy disk control. The power management features are transparent to any application software. There are two versions of 82078 floppy disk controllers, the 82078SL and 82078-1.

The 82078 is fabricated with Intel's advanced CHMOS III technology and is also available in a 44-lead QFP package.

*Other brands and names are the property of their respective owner.

Refer to the 1994 Peripheral Components Handbook for the complete data sheet on this device.

October 1993
Order Number: 290475-003

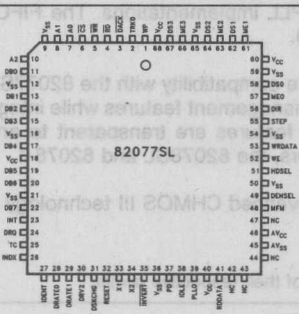
82077SL
CHMOS SINGLE-CHIP FLOPPY DISK CONTROLLER

- **Completely Compatible with Industry Standard 82077AA**
- **Single-Chip Laptop Desktop Floppy Disk Controller Solution**
  - **— 100% PC AT* Compatible**
  - **— 100% PS/2* Compatible**
  - **— 100% PS/2 Model 30 Compatible**
  - **— Fully Compatible with Intel's 386SL Microprocessor SuperSet**
  - **— Integrated Drive and Data Bus Buffers**
- **Power Management Features**
  - **— Application Software Transparency**
  - **— Programmable Powerdown Command**
  - **— Auto Powerdown and Wakeup Modes**
  - **— Two External Power Management Pins**
  - **— Typical Power Consumption in Power Down: 10 μA**
- **High Speed Processor Interface**
- **Integrated Analog Data Separator**
  - **— 250 Kbits/sec**
  - **— 300 Kbits/sec**
  - **— 500 Kbits/sec**
  - **— 1 Mbits/sec**
- **Programmable Crystal Oscillator for On or Off**
- **Integrated Tape Drive Support**
- **Perpendicular Recording Support**
- **12 mA Host Interface Drivers, 40 mA Disk Drivers**
- **Four Fully Decoded Drive Select and Motor Signals**
- **Programmable Write Precompensation Delays**
- **Addresses 256 Tracks Directly, Supports Unlimited Tracks**
- **16 Byte FIFO**
- **68-Pin PLCC**
  (See Packaging Handbook Order Number #240800, Package Type N)

The 82077SL, a 24 MHz crystal, a resistor package, and a device chip select implements a complete laptop solution. All programmable options default to 82077AA compatible values. The dual PLL data separator has better performance than most board level/discrete PLL implementations. The FIFO allows better system performance in multi-master systems (e.g., Microchannel, EISA).

The 82077SL is a superset of 82077AA. The 82077SL incorporates power management features while maintaining complete compatibility with the 82077AA/8272A floppy disk controllers. It contains programmable power management features while integrating all of the logic required for floppy disk control. The power management features are transparent to any application software. The 82077SL is available in three versions—82077SL-5, 82077SL and 82077SL-1. 82077SL-1 has all features listed in this data sheet. It supports both tape drives and 4 MB floppy drives. The 82077SL supports 4 MB floppy drives and is capable of operation at all data rates through 1 Mbps. The 82077SL-5 supports 500/300/250 Kbps data rates for high and low density floppy drives.

The 82077SL is fabricated with Intel's advanced CHMOS III technology and is available in a 68-lead PLCC (plastic) package.



290410-1

Figure 1. 82077SL Pinout

*PS/2 and PC AT are trademarks of IBM.

Refer to the 1994 Peripheral Components Handbook for the complete data sheet on this device.

October 1993
Order Number: 290410-004

89C024FT
V.42/42bis MODEM CHIP SET

- CHMOS for Low Operating Power
- Low Standby Power
- Minimum Chip Count for Small Size
- V.42 Compliant Error Correction (LAPM and *MNP4)
- V.42bis and MNP5 Data Compression Increase Throughput up to 4 Times with DTE Rates of 9600
- AT Command Set
- V.22bis, V.22 A/B, V.21, Bell 212A, and Bell 103 Compatible
- Automatically Detects Remote Modem Type and Data Rate
- On-Chip Hybrid
- DTMF and Pulse Dialing
- On-Chip Serial Port and Handshake Signals for RS-232/V.24 Interface
- Serial Interface to External NVRAM
- Automatic Speed Matching in Reliable and Normal Modes
- Hardware and Software Flow Control
- Analog/Digital Loopback Diagnostics
- Synchronous Modes
- Easily Customized Command Set and Features
- Intel's V.42/42bis and MNP Software Co-Developed with R. Scott Associates**
- Packaging:
  - — For Packages
    QN89026FT SV901 68-Pin PLCC
    QP89027 28-Pin PDIP
    Order Kit #89C024FT SZ502
  - — For Packages
    QN89026FT SV901 68-Pin PLCC
    QN89027 28-Pin PLCC
    Order Kit #89C024FT SZ503

(See Packaging Spec. Order #240800-001)

4

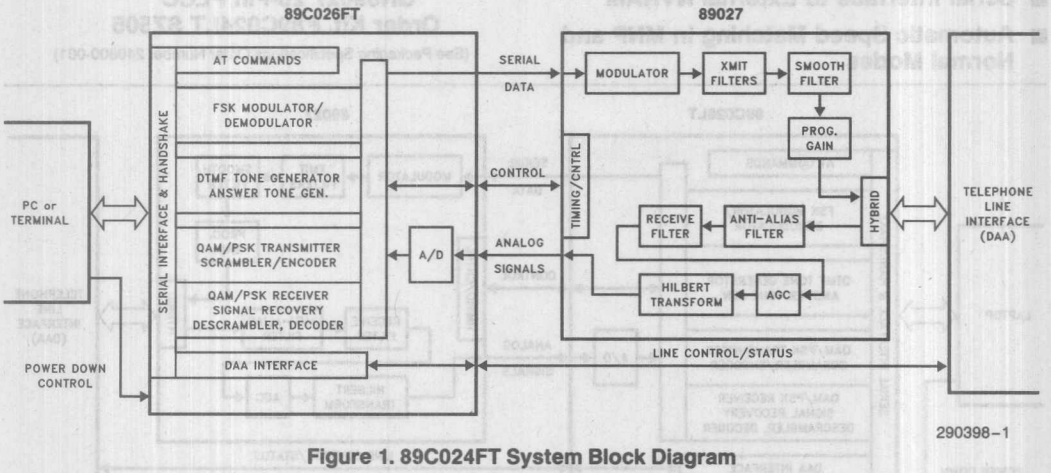


Figure 1. 89C024FT System Block Diagram

This product is not licensed by Hayes Microcomputer Products, Inc. ("Hayes") or International Business Machines ("IBM"). Specific applications of this product may be determined by Hayes or IBM to require a license. Intel Corporation does not assure any liability or provide patent identification based on such determination.

*MNP is a registered trademark of Microcom, Inc.

**R. Scott Associates, Inc., 5711 Six Forks Road, Suite 301, Raleigh, North Carolina 27609, (919) 846-7171

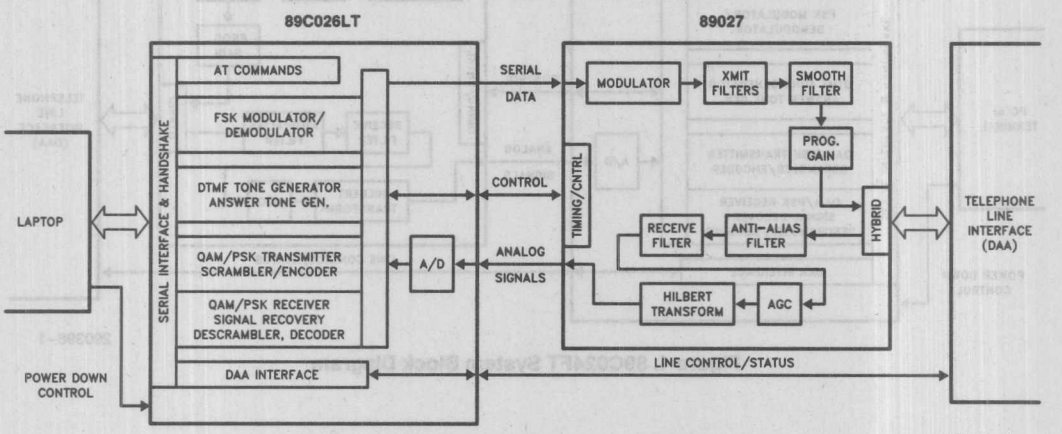
For the complete data sheet on this product, refer to the 1994 Connectivity handbook.

October 1993
Order Number: 290398-005

89C024LT
ERROR CORRECTING LAPTOP MODEM CHIP SET

- CHMOS for Low Operating Power
- Low Standby Power Requirements
- Minimum Chip Count for Small size
- MNP* Operation through Class 4 for Error Correction
- MNP Class 5 Data Compression for Increased Throughput
- AT Command Set
- V.22 bis, V.22 A/B, V.21, Bell 212A, and Bell 103 Compatible
- For Public Switched Telephone Network and Unconditioned Leased Line Applications
- Automatically Detects Remote Modem Type and Data Rate
- On-Chip Hybrid
- DTMF and Pulse Dialing
- On-Chip Serial Port and Handshake Signals for RS-232/V.24 Interface
- Serial Interface to External NVRAM
- Automatic Speed Matching in MNP and Normal Modes
- Hardware and Software Flow Control
- Analog/Digital Loopback Diagnostics
- Telephone Line Audio Monitor Output
- Full Set of Control Signals for DAA Interface
- International Call Progress Tone Detection Capabilities
- Automatic Adaptive Equalization
- Synchronous Modes
- Easily Customized Command Set and Features
- Intel's MNP Software Co-Developed with R. Scott Associates**
- Packaging:
  - — For Packages
    QN89026LT SV782 68-Pin PLCC
    QN89027 28-Pin PDIP
    Order Kit #89C024LT SZ504
  - — For Packages
    QN89026LT SV782 68-Pin PLCC
    QN89027 28-Pin PLCC
    Order Kit #89C024LT SZ505

(See Packaging Specifications Order Number 240800-001)



290272-1

Figure 1. 89C024LT System Block Diagram

This product is not licensed by Hayes Microcomputer Products, Inc. ("Hayes"). Specific applications of this product may be determined by Hayes to require a license. Intel Corporation does not assume any liability or provide patent indemnification based on such determination.

*MNP is a registered trademark of Microcom, Inc.

**R. Scott Associates, Inc., 5711 Six Forks Road, Suite 301, Raleigh, North Carolina 27609, (919) 846-7171

For the complete data sheet on this product, refer to the 1994 Connectivity handbook.

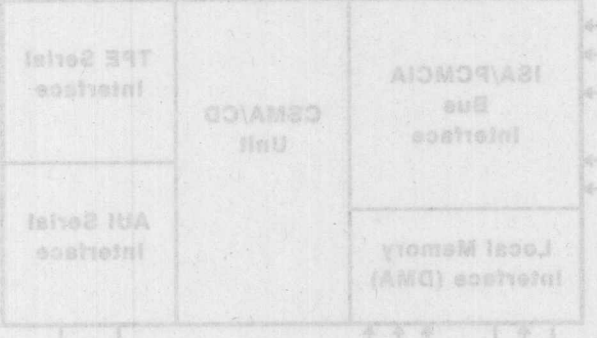
October 1993
Order Number: 290272-005

89C124FX
DATA/FAX MODEM CHIP SET

- 9600 bps Send and Receive FAX
- V.29 and V.27ter Compatible
- Supports Communicating Applications Specification (CAS)
- EIA/TIA-578 Compliant FAX Command Set (Service Class 1)
- Compatible with CCITT Group 3 FAX Machines
- 2400 bps Data Modem
- V.22bis, V.22 A/B, V.21, Bell 212A, and Bell 103 Compatible
- V.42 Compliant Error Correction (LAPM and *MNP4)
- V.42bis and MNP5 Data Compression
- Easy Upgrade from Intel Modem Chip Sets
- AT Command Set
- Minimum Chip Count for Small Size
- CHMOS for Low Operating Power
- Low Standby Power
- On-Chip Hybrid
- DTMF and Pulse Dialing
- Automatic Speed Matching in Reliable and Normal Modes
- Serial Interface to External NVRAM
- Hardware and Software Flow Control
- Analog/Digital Loopback Diagnostics
- Automatically Detects Remote Modem Type and Data Rate
- Easily Customized Command Set and Features
- Synchronous Modes
- On-Chip Serial Port Handshake Signals for RS-232/V.24 Interface
- Packaging
  - — 89127:
    28-Lead PDIP and PLCC, 64-Lead QFP Packages
    Package Type P, N and S
  - — 89C126FX:
    68-Lead PLCC and 80-Lead QFP
    Package Type N and SB

(See Packaging Spec Order No. 240800)

4



This product is not licensed by Hayes Microcomputer Products, Inc. ("Hayes") or International Business Machines ("IBM"). Specific applications of this product may be determined by Hayes or IBM to require a license. Intel Corporation does not assume any liability or provide patent indemnification based on such determination.

*MNP is a registered trademark of Microcom, Inc.

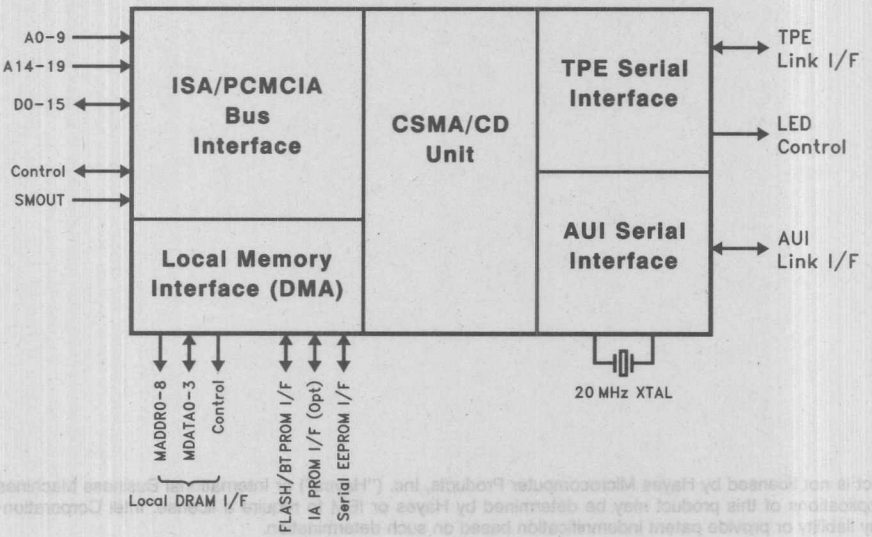
**R. Scott Associates, Inc., 7701 Six Forks Rd, Suite 120, Raleigh, North Carolina 27615, (919) 846-7171

For the complete data sheet on this product, refer to the 1994 Connectivity handbook.

October 1992
Order Number: 290415-002

82595
ISA/PCMCIA HIGH INTEGRATION ETHERNET CONTROLLER

- **Optimal Integration for Lowest Cost Solution**
  - — Glueless 8-Bit/16-Bit ISA/PCMCIA 2.0 Bus Interface
  - — Provides Fully 802.3 Compliant AUI and TPE Serial Interface
  - — Local DRAM Support up to 64 Kbytes
  - — FLASH/EPROM Boot Support
  - — Hardware and Software Portable between Motherboard, Adapter, and PCMCIA IO Card Solution
- **High Performance Networking Functions**
  - — 16-Bit IO Accesses to Local DRAM with Zero Added Wait-States
  - — Ring Buffer Structure for Continuous Frame Reception and Transmit Chaining
  - — Automatic Retransmission on Collision
  - — Automatically Corrects TPE Polarity Switching Problems
- **Low Power CHMOS IV Technology**
- **Ease of Use**
  - — Design Time Reduced by High Integration
  - — EEPROM Interface to Support Jumperless Design
  - — Software Structures Optimized to Reduce Processing Steps
  - — Automatically Maps into Unused PC IO Location to Help Eliminate LAN Setup Problems
  - — All Software Structures Contained in One 16-Byte IO Space
  - — Automatic or Manual Switching between TPE and AUI Ports
  - — JTAG Port for Reduced Board Testing Times
- **Power Management**
  - — SL Compatible $\overline{\text{SMOUT}}$ Power Down Input
  - — Software Power Down Command for non-SL Systems
- **144-Lead tQFP Package Provides Smallest Available Form Factor**
  (See Packaging Spec., Order No. 240800)



290458-1

Figure 1. 82595 Block Diagram

For the complete data sheet on this product, refer to the 1994 Networking handbook.

October 1993
Order Number: 290458-003

82593
CSMA/CD CORE LAN CONTROLLER

- **Supports Industry Standard LANs**
  - IEEE 10BASE5 (Ethernet*)
  - IEEE 10BASE2 (Cheapernet)
  - IEEE 10BASE-T (TPE)
- **Simple, High-Performance Control and Data Interface**
  - Control and Status via $\overline{RD}$, $\overline{WR}$, and $\overline{CS}$ Lines
  - Data Transfers via DMA Interface
  - Two Clocks per DMA Transfer
  - Programmable Bus Throttle Timer
- **High-Performance Networking Functions**
  - Automatic Retransmission from Internal FIFO
  - Back-to-Back Frame Reception with No CPU Intervention
  - Receive Ring Buffer Memory Structure
  - Transmit Frame Chaining
  - 96-Byte Transmit FIFO and 96-Byte Receive FIFO
- **High Speed, 5V CHMOS IV (P648.8) Technology**
- **Serial Bit Rates up to 20 Mb/s (82593SX)**
  - Direct Interface to Intel 82C501AD ESI or AMD 7992 SIA
  - Conforms to 802.3 CSMA/CD Standard
- **On-Board Diagnostics**
  - Internal and External Loopback Operation
  - Internal Register Dump
  - TDR Functionality
- **44-Lead PLCC Package Type N (82593SX), 44-Lead QFP Package Type S (82593SX) or 28-Pin PDIP**
  - 82593SX (8/16-Bit) System Clock up to 20 MHz
  - 82593SX Package Pin Compatible with Intel 82592 PLCC

(See Packaging Spec., Order No. 240800-001 Package Type N, S and P)

4

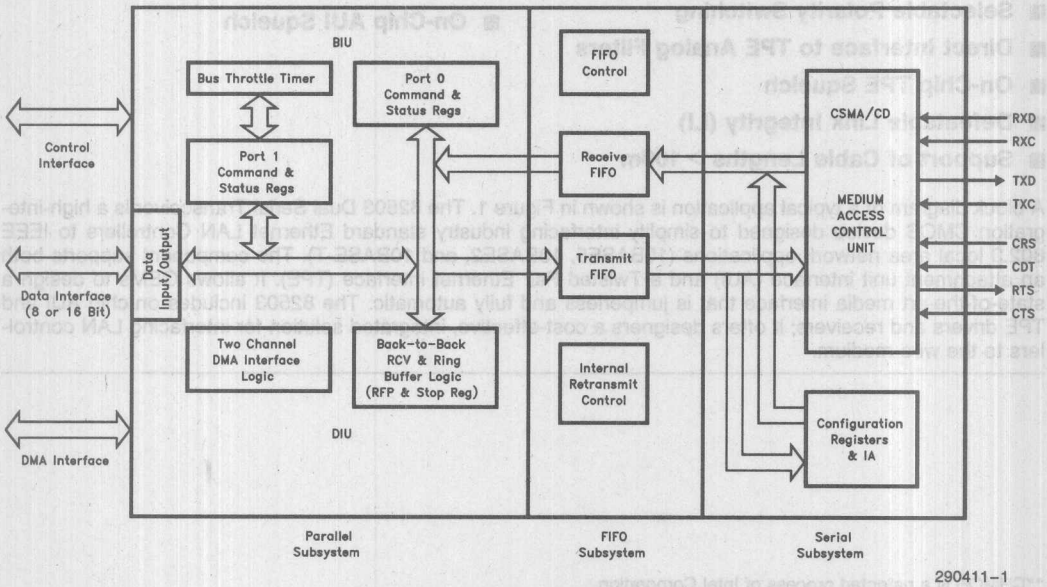


Figure 1. 82593 Block Diagram

*Ethernet is a registered trademark of Xerox Corporation.

For the complete data sheet on this product, refer to the 1994 Networking handbook.

October 1992
Order Number: 290411-004

82503
DUAL SERIAL TRANSCEIVER (DST)

82503 PRODUCT FEATURE SET OVERVIEW

- Single Component Ethernet* Interface to Both 802.3 10BASE-T and AUI
- Automatic or Manual Port Selection
- Manchester Encoder/Decoder and Clock Recovery
- No Glue Interface to Industry-Standard LAN Controllers
  - — Intel 82586, 82590, 82593 and 82596
  - — AMD 7990 (LANCE*)
  - — National Semiconductor 8390 and 83932 (SONIC*)
  - — Western Digital 83C690
  - — Fujitsu 86950 (Etherstar*)
- Diagnostic Loopback
- Reset, Low Power Modes
- Network Status Indicators
- Defeatable Jabber Timer
- User Test Modes
- 10 MHz Transmit Clock Generator
- One Micron CHMOS** IV (Px48) Technology
- Single 5-V Supply

INTERFACE FEATURES

TPE

- Complies with 10BASE-T, IEEE Std. 802.3i-1990 for Twisted Pair Ethernet
- Selectable Polarity Switching
- Direct Interface to TPE Analog Filters
- On-Chip TPE Squelch
- Defeatable Link Integrity (LI)
- Support of Cable Lengths > 100m

AUI

- Complies with IEEE 802.3 AUI Standard
- Direct Interface to AUI Transformers
- On-Chip AUI Squelch

A block diagram of a typical application is shown in Figure 1. The 82503 Dual Serial Transceiver is a high-integration CMOS device designed to simplify interfacing industry standard Ethernet LAN Controllers to IEEE 802.3 local area network applications (10BASE5, 10BASE2, and 10BASE-T). The component supports both an attachment unit interface (AUI) and a Twisted Pair Ethernet interface (TPE). It allows OEMs to design a state-of-the-art media interface that is jumperless and fully automatic. The 82503 includes on-chip AUI and TPE drivers and receivers; it offers designers a cost-effective, integrated solution for interfacing LAN controllers to the wire medium.

**CHMOS is a patented process of Intel Corporation.
*Ethernet is a registered trademark of Xerox Corporation.
LANCE is a registered trademark of Advanced Micro Devices.
Etherstar is a registered trademark of Fujitsu Electronics.
Sonic is a registered trademark of National Semiconductor Corporation.

For the complete data sheet on this product, refer to the 1994 Networking handbook.

November 1992
Order Number: 290421-003

Ethernet* LAN Card Product Brief

Product Highlights

- Complete plug and play PCMCIA LAN solution. Comes with all drivers, installation, card and socket services and card management software necessary for reliable operation in a PCMCIA slot.
- Drivers for all major network operating systems.
- Industry-recognized Intel SoftSet installation software. Easy to operate and manage.
- Based on highly integrated Intel 82595 Ethernet Controller
- Complies with PCMCIA 2.0/ JEIDA 4.1 68-pin standard.
- 5 mm-thick PCMCIA Type II card.
- Detachable line adapter module (LAM) for multiple media attachment.
- Ethernet IEEE 802.3 compatibility (10BASE-T/TPE, 10BASE- 2/BNC).
- Activity and link integrity LEDs.



The Intel PCMCIA Ethernet LAN Card brings high performance and ease of use to PCMCIA networking. It lets you put networking capabilities into laptop computers without a lot of hassle, headache or expense.

It's a plug and play solution, providing PCMCIA network- readiness right out of the box. All the software you need comes with the card: drivers for the most popular network operating systems (Novell NetWare 2.2, 3.11, 4.0 and Lite 1.0; Microsoft* LAN Manager* 2.X; IBM* LAN Server* 2.X; Banyan Vines* 4.x; and Microsoft Windows for Workgroup* 3.1), PCMCIA- compliant card and socket services from SystemSoft*, and Intel's own card manager and Softset auto-configuration, auto-installation software that gets users on the network fast.*

The Intel PCMCIA Ethernet LAN Card is based on the highly integrated Intel 82595 ISA/PCMCIA Ethernet Controller, giving you 16-bit desktop performance in a PCMCIA form factor. As we develop additional software functionality for our 82595 line, you'll be able to leverage it across your entire 82595-based product line, from chips to NICs to PCMCIA form factor products. You decrease your time to market by taking advantage of Intel's product development efforts, while increasing the value of your products.

The card is standard PCMCIA 2.0 68-pin form factor and only 5 mm thick. It's passed Intel's extensive quality and reliability testing to ensure that it stands up to the rigors of mobile users on the go. These include PCMCIA mechanical qualification testing such as torque, bend, shock, vibration and environmental testing across extreme temperatures and voltages. Our CMOS technology and highly integrated 82595 are very power efficient, letting mobile users stay connected to the network for a long time. Maximum power draw is 85 mA; in idle mode, power consumption drops to 20 mA.

Finally, we've made it easy to customize our card and its accessories to your own OEM marketing needs. Manuals, software diskettes and the cards themselves can all be tailored to reflect your company's look.

Product Description

The Intel PCMCIA Ethernet LAN Card is the fastest, easiest way to deliver network-ready laptops to your customers. It snaps in and installs in minutes, not hours, thanks to a disk-full of software to make your job easier. Our industry-standard SoftSet installation utility automatically configures the card and sets up the software with a single command. Built-in card and socket services software provides card recognition and compatibility. Built-in card management software performs IRQ management and allows the card to be installed even when the system is running, so you don't have to reboot. There's even built-in driver support for popular network operating systems from Novell, Microsoft, IBM and Banyan. There are no jumpers or switches to set, no IRQ addresses to labor over. The card's installed and working in five minutes. Once installed, it's easy to operate too, improving customer satisfaction and decreasing the number of support calls you receive.

The card is fully PCMCIA 2.0/JEIDA 4.1 compliant with a standard 68-pin form factor. It's also fully compliant with Ethernet IEEE 802.3 standard for 10BASE-T and 10BASE-2 wiring.

There's a detachable line adapter module (LAM) for attachment to multiple media and LEDs to indicate active and link integrity.

Features	Benefits
— Installation, card manager, and card and socket services software	— A complete solution; no need for any other pieces Easy to install, easy to configure, easy to use
— Broad client driver support	— Meets broad target market. Usable on industry-standard networks like Novell, LAN Manager and Banyan
— Complies with PCMCIA 2.0/JEIDA 4.1 standards	— No jumpers Easy to transport Small form factor
— Intel-based 82595	— Great performance; 8 & 16-bit data path Glueless interface to PCMCIA Bus
— Activity and link integrity LEDs	— Indicates card status, improves diagnostics
— Supports both twisted pair Ethernet (10BASE-T) and ThinCoax (BNC 10BASE-2)	— Flexibility to connect to multiple network media

Product Codes

MBLA8110	TPE all geographies
MBLA8120US	BNC US
MBLA8120EU	BNC Europe

Additional Literature

Local Area Networking Family Product Brief	297085-002
EtherExpress Family Brochure	D413.01
TokenExpress Family Brochure	D414.02
Intel 82595 Data Sheet	290458-003
Intel 82595 User's Manual	TBD

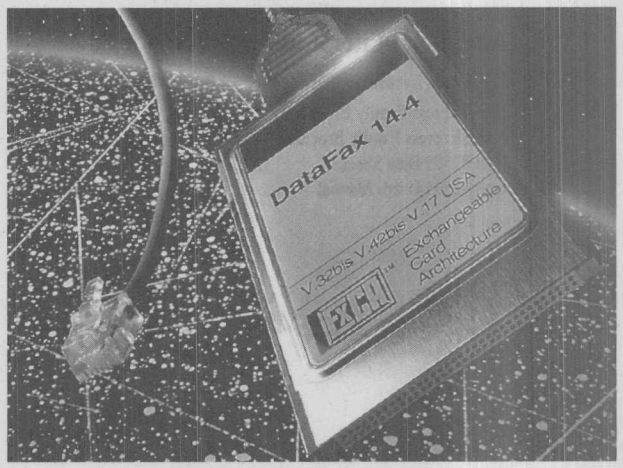
* Other brands and names are the property of their respective owners.

Order Number: 297120-003
©Intel Corporation, 1993

DataFax 14.4 Card Product Brief

Product Highlights

- LAM-less design (Integrated DAA)
- Complies with PCMCIA 2.0 and JEIDA 4.1 standards
- PCMCIA Type II card — 5.0 mm thick
- Automatic power down mode
- DTMF AND PULSE dialing
- Fully compliant with CCITT T.30 and T.4 (Group 3 fax)
- Compliant with CCITT V.17 (14.4 Kbps fax)
- Provides both Send and Receive fax capability
- 8-bit I/O bus interface
- Includes Ring Detect notification to host computer in the power down mode
- FCC Class B, UL and DOC/UL (Canada)
- Compatible with EIA/TIA 602 (AT command set)
- Data modem complies with CCITT V.32bis, V.32, V.22bis, V.22, V.21, Bell* 212a, 103
- Supports CCITT V.42 error detection and V.42bis data compression
- Provides MNP/5* data compression for backward compatibility
- Requires no external power
- MNP/10
- Hayes AutoSync



With the design of the DataFax 14.4, Intel introduces a high speed fax card with no external line adapter module. Lightweight. Easy to carry. And less chance of damage or loss. Not only does the DataFax 14.4 feature integrated DAA, but its high speed transmission helps your customers reduce the telephone expenses associated with faxes and modems.

The DataFax 14.4 fax card is also Group 3 compliant, assuring worldwide compatibility with most fax machines operating today. It sends and receives faxes and transfers files to or from notebook or hand-held computers over the public telephone network.

The Intel DataFax 14.4 also conforms to PCMCIA 2.0 AND JEIDA 4.1 physical and electrical standards for portable computers. The size of a credit card, the DataFax 14.4 slides into an external slot in the notebook computer, and connects easily into the public switched telephone network. The card features four power management modes — on-line, active, power save, and power down — to ensure the lowest possible power consumption. No external power is required.

Product Description

The DataFax 14.4 combines high speed with an integrated DAA design to provide optimal connectivity for notebooks and sub-notebooks. PCMCIA cards enhance the multiple functionality of the notebook computer. Intel's integrated DAA design incorporates the telephone interface circuitry directly onto the card. This includes a ring detector and telephone line coupling transformer. A six-foot cable connects the DataFax 14.4 card to a standard RJ-11 modular telephone jack for connectivity to the telephone network.

The DataFax 14.4 supports V.42 error correction which ensures that errors caused by the phone system are automatically corrected. The DataFax 14.4 increases data throughput with V.42bis data compression. This detects redundant characters, character sequences, and uses fewer bits to send more frequently occurring sequences. DataFax 14.4 users enjoy an effective throughput of up to 57,600 bits per second.

Unlike other card manufacturers, Intel provides a full-service program for card labeling, custom kitting with third party applications, and fulfillment. The Intel DataFax 14.4 is the only fax card designed with recessed covers in order to accept adhesive labels, front and back, insuring quick turnaround for custom orders.

Features	Benefits
— Integrated DAA	— No external circuitry — Compact and lightweight — Easy to carry — Fits in briefcase with notebook computer
— Exchangeable with other cards	— Single slot serves multiple functions
— CCITT V.32bis (14.4 Kbps)	— Ensures connectivity worldwide — Faster transmission means less costly phone bills
— Supports V.42/V.42bis	— Provides high throughput (57.6 Kbps)
— MNP/10	— Enhanced data throughput with cellular connections (required phone adapter)
— Hayes AutoSync	— Provides synchronous communications capability (software required)

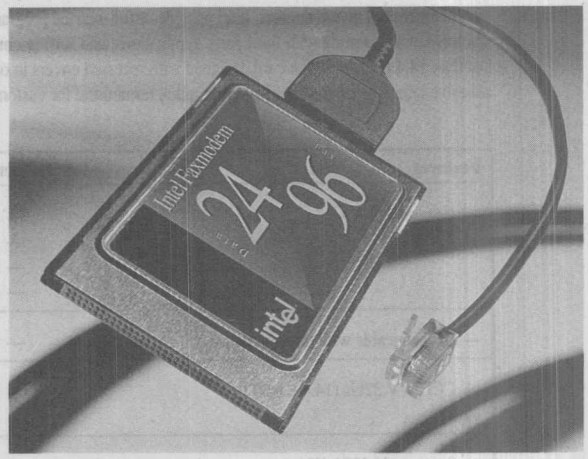
*Other brands and names are the property of their respective owners.

Order Number: 297311-002
© Intel Corporation, 1993

Faxmodem 24/96 Card Product Brief

Product Highlights

- Integrated DAA on the card for U.S./Canada
- Complies with PCMCIA 2.0 and JEIDA 4.1 standards
- PCMCIA Type II card
  - 5mm thick
- Fully compliant with CCITT T.30 and T.4 (group 3 fax)
- Provides both Send and Receive fax capability
- Fully compliant with EIA/TIA 578 (fax class 1 command set)
- Data modem complies with CCITT V.22bis, V.22, V.21, Bell* 212a and 103
- Supports CCITT V.42 error detection and V.42bis data compression
- Provides MNP5 data compression for backward compatibility
- Includes Ring Detect notification to host computer in the power down mode
- Multiple power conservation modes
- Requires no external power



The Faxmodem 24/96 enables you to send or receive faxes and transfer files to or from notebook computers over the public telephone network. It is the latest member of Intel's I/O card "plug and play" family.

The Intel Faxmodem 24/96 also conforms to PCMCIA 2.0 AND JEIDA 4.1 physical and electrical standards for portable computers. The card's light weight and low power consumption combine to provide high-performance, low-cost connectivity for notebooks and sub-notebooks.

Approximately the size of a credit card, the Faxmodem 24/96 slides into an external slot in the notebook computer, connecting easily into the public switched telephone network. The card features four power management modes — on-line, active, power save, and power down — to ensure the lowest possible power consumption. No external power is required.

Product Description

The Faxmodem 24/96 card provides convenient, high-performance mobile fax and data communications capability for the notebook computer user. Adherence to accepted national and international standards provides the user with worldwide connectivity for both fax (approximately 30 million machines in use worldwide) and data transfer (V.22bis is the most widely used communications standard in the world). Its exchangeability with other PCMCIA cards enhances the multiple functionality of the notebook computer.

intel

The Faxmodem 24/96 card contains the Intel 89C124FX integrated data-fax modem chipset, a UART, a microcontroller, an analog front-end and other supporting devices. The 16450-type UART provides an 8-bit data bus interface. This, together with the Class 1 AT command set support, provides compatibility with all the major fax application software.

For U.S. and Canada, the DAA is integrated on the card. For rest of world, the detachable line adapter module incorporates country-specific telephone interface circuitry. This consists of the ring detector and telephone line coupling transformer. A standard RJ-11 modular telephone jack provides connectivity to the telephone network. A short cable connects the line adapter module to the Faxmodem 24/96 card.

The Faxmodem 24/96 supports V.42 error correction which ensures that errors caused by the phone system are automatically corrected. The Faxmodem 24/96 also supports V.42bis data compression. This increases data throughput by detecting redundant characters, character sequences, and using fewer bits to send more frequently occurring sequences. Throughput for file transfer operations is increased by as much as 400 percent, providing the user with an effective 9600 bits per second.

Features	Benefits
— Integrated DAA	— No external circuitry — Compact and lightweight — Easy to carry — Fits in briefcase with notebook computer
— Exchangeable with other cards	— Single slot serves multiple functions
— Group 3 fax	— Compatible with worldwide installed base of fax machines
— Class 1 fax command set	— Compatible with standard communications packages
— CCITT V.22bis, V.22, V.21, Bell 212 and 103	— Ensures connectivity worldwide
— Supports V.42/V.42bis	— Faster; up to 4 to 1 data compression providing an effective throughput of 9600 bits per second for file transfers
— Modem supports AT command set	— Compatible with standard communications packages
— Multiple power conservation modes	— Prolong system battery life
— Factory Configuration Option for Cellular Network	— Ease of use

* Other brands and names are the property of their respective owners.

Order Number: 297390-001
©Intel Corporation, 1993

85C220/85C224-100, -80 AND -66 FAST REGISTERED SPEED T_{SU}, T_{SO} 8-MACROCELL PLDs

These register optimized timing PLDs offer superior design features:

- **Low-Power, High-Performance Upgrade for SSI/MSI Logic and Bipolar PALs* High-Performance Systems**
- **Replacement or Upgrade for 16V8/ 20V8 PAL and GAL Architecture**
- **8 Macrocells with Independently Programmable I/O Architecture**
- **Up to 18 Inputs (10 Dedicated and 8 I/O) and 8 Outputs**
- **Programmable "Security Bit" Allows Total Protection of Proprietary Designs**
- **100% Generally Tested Logic Array**

85C220-100 AND 85C224-100

- **100 MHz Max Frequency (External Feedback); 5.5 ns (Max) Clock to Output; 4.5 ns (Min) Set-Up Time**
- **Meets Critical Timing Requirements of Advanced Intel Microprocessor Systems**
- **7.5 ns (Max) Propagation Delay**
- **Typical I_{CC} = 90 mA**
- **Available in 20-Pin and 28-Pin PLCC Packages**
- **Low Power and Output Skew for Clock Device Applications**

85C220-80 AND 85C224-80

- **Quarter Power (I_{CC} = 40 mA); Programmable Zero Power Mode (50 μA Typical)**
- **80 MHz Max Frequency (External Feedback); 5.5 ns (Max) Clock to Output; 7 ns (Min) Set-Up Time**
- **10 ns Propagation Delay**
- **High-Speed Upgrade to EP320, EP330, and 5C032**
- **Available in 300-mil 20-Pin and 24-Pin CerDIP/PDIP Packages, and 20-Pin and 28-Pin PLCC Packages**

Intel386™, i486™ and i860™ are trademarks of Intel Corporation.
*PAL is a registered trademark of Advanced Micro Devices.

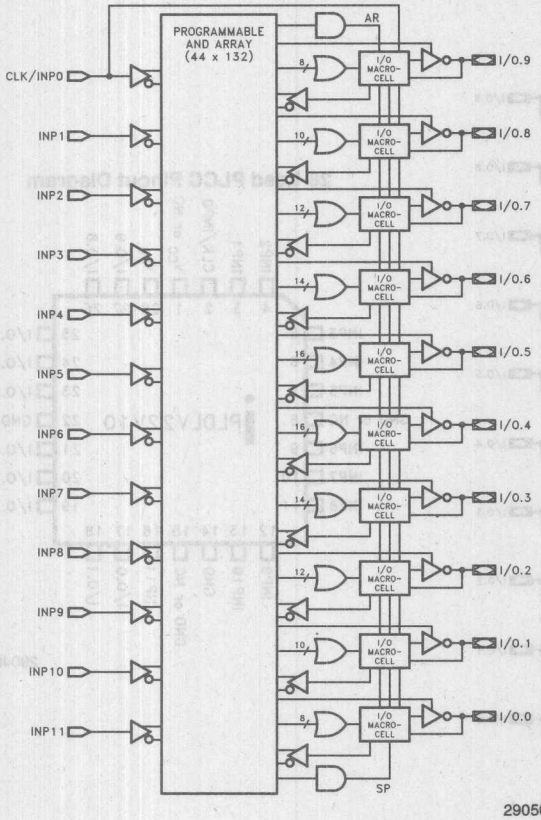
Refer to the 1994 Programmable Logic Handbook for the complete data sheet on this device.

October 1993
Order Number: 290224-005

iPLD22V10L-5
LOW POWER, ELECTRICALLY ERASABLE 10-MACROCELL CMOS PLD

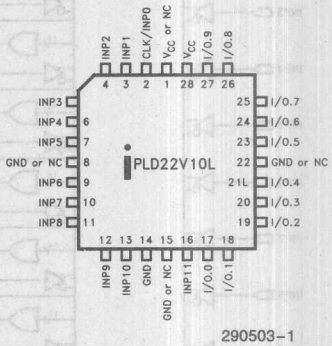
- Low Power, High Speed Upgrade to BiCMOS/Bipolar 22V10 and CMOS Equivalents
- Supply Voltage Range 4.75V to 5.25V
- t_{PD} 5 ns, 142 MHz with Feedback
- Maximum I_{SB} = 5 mA
- Typical I_{CC} = 15 mA @100 MHz
- 12 Dedicated Inputs and 10 I/O Pins
- 10 Macrocells with Programmable I/O Architecture (Registered/ Combinatorial)
- Sub-micron CHMOS III Electrical Erasable Technology
- Programmable "Security Bit" Allows Total Protection of Proprietary Designs
- 100% Generically Tested Logic Array
- Available in 28-Pin PLCC Packages

iPLD22V10 Global Architecture



290503-2

28-lead PLCC Pinout Diagram



290503-1

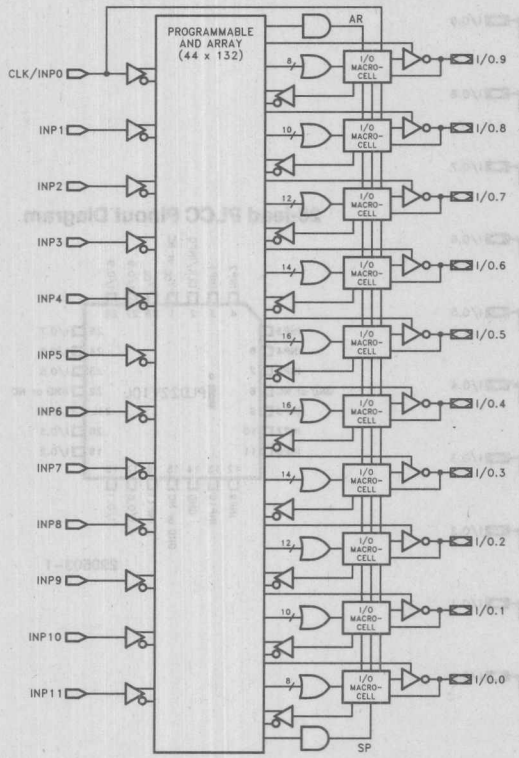
Refer to the 1994 Programmable Logic Handbook for the complete data sheet on this device.

November 1993
Order Number: 290503-001

LOW VOLTAGE, ELECTRICALLY ERASABLE 10-MACROCELL CMOS PLD

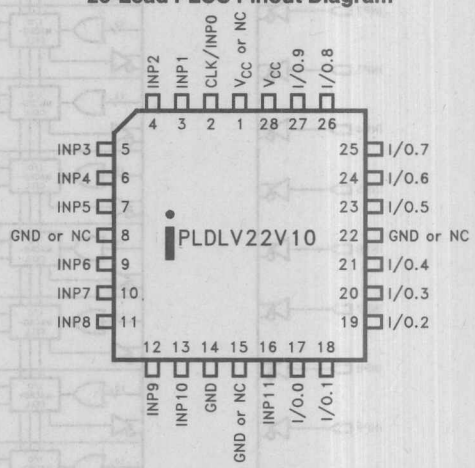
- Low Voltage, Low Current, High Speed Upgrade to BiCMOS/Bipolar 22V10 and CMOS Equivalents
- Supply Voltage Range 3.0V to 3.6V. Ideal for Battery Powered Systems
- t_{PD} 5 ns, 142 MHz with Feedback
- Maximum I_{SB} = 100 μA
- Typical I_{CC} = 10 mA @ 100 MHz
- 12 Dedicated Inputs, 10 I/O Pins
- 10 Macrocells with Programmable I/O Architecture (Registered/ Combinatorial)
- Global Asynchronous Clear and Synchronous Preset P-terms
- Sub-micron CHMOS III Electrical Erasable Technology
- Programmable "Security Bit" Allows Total Protection of Proprietary Designs
- Available in 28-Pin PLCC Packages

iPLDLV22V10 Global Architecture



290495-2

28-Lead PLCC Pinout Diagram



290495-1

Refer to the 1994 Programmable Logic Handbook for the complete data sheet on this device.

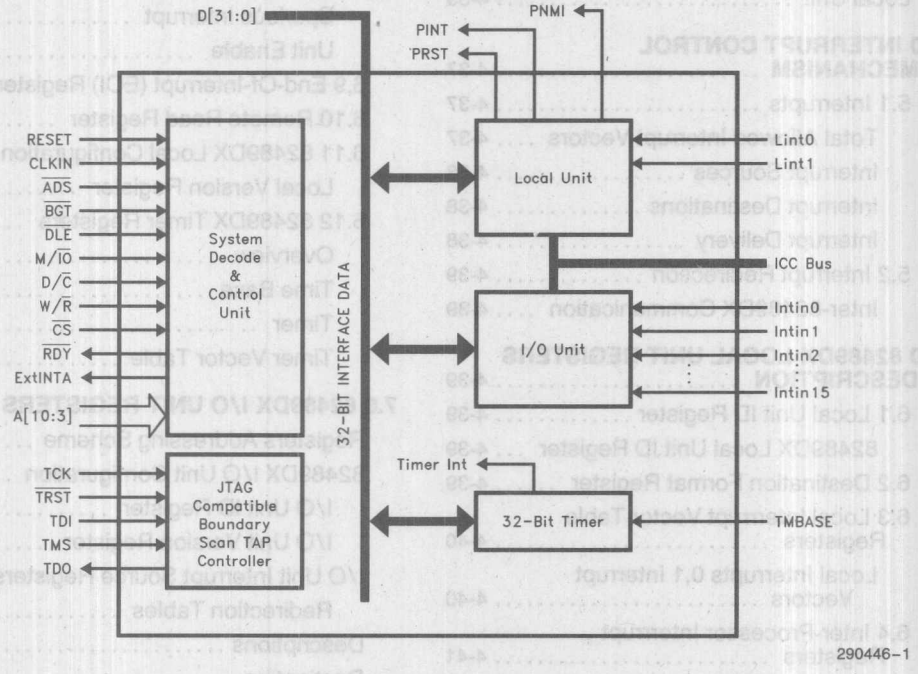
October 1993
Order Number: 290495-001

82489DX ADVANCED PROGRAMMABLE INTERRUPT CONTROLLER

82489DX FEATURES OVERVIEW

- Advanced Interrupt Controller for 32-Bit Operating Systems
- Solution for Multiprocessor Interrupt Management
- Dynamic Interrupt Distribution for Load Balancing in MP Systems
- Separate Nibble Bus (Interrupt Controller Communications (ICC) Bus) for Interrupt Messages
- Inter-Processor Interrupts
- Various Addressing Schemes—Broadcast, Fixed, Lowest Priority, etc.
- Compatibility Mode with 8259A
- 32-Bit Internal Registers
- Integrated Timer Support
- 33 MHz Operation
- 132-Lead PQFP Package, Package Type KU
  (See Packaging Specification. Order Number: 240800)

82489DX Block Diagram



290446-1

Refer to Application Note AP-388: 82489DX User's Manual (Order Number 292116) when evaluating your design needs.

82489DX
Advanced Programmable Interrupt Controller

CONTENTS

PAGE

1.0 INTRODUCTION 4-29

2.0 FUNCTIONAL OVERVIEW 4-30

ICC Bus 4-30

Local Unit 4-30

I/O Unit 4-30

Timer 4-30

3.0 PIN DESCRIPTION 4-30

4.0 FUNCTIONAL DESCRIPTION 4-34

I/O Unit 4-34

Local Unit 4-35

5.0 INTERRUPT CONTROL MECHANISM 4-37

5.1 Interrupts 4-37

Total Allowed Interrupt Vectors 4-37

Interrupt Sources 4-38

Interrupt Destinations 4-38

Interrupt Delivery 4-38

5.2 Interrupt Redirection 4-39

Inter-82489DX Communication 4-39

6.0 82489DX LOCAL UNIT REGISTERS DESCRIPTION 4-39

6.1 Local Unit ID Register 4-39

82489DX Local Unit ID Register ... 4-39

6.2 Destination Format Register 4-39

6.3 Local Interrrupt Vector Table Registers 4-40

Local Interrupts 0,1 Interrupt Vectors 4-40

6.4 Inter-Processor Interrrupt Registers 4-41

Interrupt Command Register [31:0] 4-41

CONTENTS

PAGE

Interrupt Command Register [63:32] 4-43

6.5 IRR, ISR, TMR Registers 4-43

Interrupt Acceptance 4-43

Acceptance Mechanism 4-45

6.6 Tracking Processor Priority 4-47

Task Priority Register 4-47

6.7 Dispensing Interrupts 4-48

Dispensing Interrupts to the Local Processor 4-48

6.8 Spurious Interrupt Vector Register 4-48

Spurious Interrupt 4-48

Unit Enable 4-48

6.9 End-Of-Interrupt (EOI) Register ... 4-48

6.10 Remote Read Register 4-49

6.11 82489DX Local Configuration 4-49

Local Version Register 4-49

6.12 82489DX Timer Registers 4-49

Overview 4-49

Time Base 4-49

Timer 4-50

Timer Vector Table 4-50

7.0 82489DX I/O UNIT REGISTERS 4-51

Registers Addressing Scheme 4-51

82489DX I/O Unit Configuration 4-52

I/O Unit ID Register 4-52

I/O Unit Version Register 4-52

I/O Unit Interrupt Source Registers ... 4-52

Redirection Tables 4-52

Descriptions 4-53

Destination 4-54

CONTENTS

PAGE

8.0 ICC BUS DEFINITION 4-55
- Physical Characteristics 4-55
- Bus Arbitration 4-55
- Lowest-Priority Arbitration 4-56
- ICC Bus Message Formats 4-56
- Long Message Format 4-58

9.0 HARDWARE TIMINGS 4-59
- Interfacing to the ICC Bus 4-60
- First Order Buffer Models 4-60
- MBO Pull-Up Register 4-60
- Driving Lumped Capacitance 4-60
- Driving Transmission Lines 4-61
- External Drivers/Buffered ICC Bus 4-63
- Transmission Line Termination 4-65
- ICC Bus Operating Frequency 4-65
- 9.1 82489DX Register Access Timing 4-67
  - Timing Diagram Notation 4-67
  - Register WRITE Timing 4-68
  - Register READ Timing 4-73
  - Interrupt Acknowledge Timing 4-73
  - Reset and Miscellaneous Timing ... 4-74

10.0 BOUNDARY SCAN DESCRIPTION 4-74
- 10.1 Boundary Scan Architecture 4-74
  - Test Access Ports 4-75
  - TAP Controller 4-75
  - Test-Logic-Reset 4-76
  - Run-Test/Idle 4-77
  - Select-DR-Scan 4-77
  - Select-IR-Scan 4-77
  - Capture-DR 4-77
  - Shift-DR 4-77
  - Exit 1-DR 4-77
  - Pause-DR 4-77
  - Exit 2-DR 4-77
  - Update-DR 4-78
  - Capture-IR 4-78
  - Shift-IR 4-78
  - Exit 1-IR 4-78

CONTENTS

PAGE

- Pause-IR 4-78
- Exit 2-IR 4-78
- Update-IR 4-78
- Instruction Register 4-79
- Bypass Instruction 4-79
- Extest Instruction 4-79
- Sample/Preload Instruction 4-79
- dcode Instruction 4-79
- Device Identification Register (DID) ... 4-79
- Boundary Scan Register 4-79
- Boundary Scan Cell Names in Order from tdi to tdo 4-81
- Bypass Register 4-83
- JTAG TAP Controller Initialization 4-83

11.0 ELECTRICAL CHARACTERISTICS 4-84
- 11.1 D.C. Specifications 4-84
- 11.2 A.C. Specifications 4-85

12.0 REGISTER SUMMARY 4-87
- I/O Unit Registers 4-88
- Local Unit Registers 4-89

13.0 TIMING DIAGRAMS 4-90

14.0 PACKAGE PIN-OUT 4-94

15.0 PACKAGE THERMAL SPECIFICATION 4-95

16.0 GUIDELINES FOR 82489DX USERS 4-96
- 16.1 Initialization 4-96
- 16.2 Compatibility 4-96
  - Compatibility Levels 4-96
  - 82489DX/8259A Interaction 4-96
  - 82489DX/8259A Dual Mode Connection 4-97
- 16.3 Hardware Guidelines 4-98
  - 82489DX Hardware State on Reset 4-98
  - Pull Up and Pull Down Resistors ... 4-98
  - Pint and ExINTA Timings 4-98
  - ExtINTA Timings 4-98

CONTENTS PAGE

82489DX and Memory Mapping ... 4-98
JTAG Circuit Considerations ... 4-98
16.4 Programming Guidelines ... 4-99
Unique ID Requirement ... 4-99
Atomic Write Read to Task Priority Register ... 4-99
Critical Regions and Mutual Exclusion ... 4-99
Interrupt Command Register Programming Sequence ... 4-99
Interrupt Vector ... 4-99
Local and I/O Unit ... 4-99
ICR (Interrupt Command Register) ... 4-99
ISR/IRR/TMR ... 4-100
Focus Processor ... 4-100
ExtINT Interrupt Posting ... 4-100
Synchronizing Arb IDs ... 4-100
Lowest Priority ... 4-100

CONTENTS PAGE

Disabling Local Unit ... 4-100
Issuing EOI ... 4-101
External Interrupts and EOI ... 4-101
Spurious Interrupts and EOI ... 4-101
NMI and EOI ... 4-101
Task Priority Register ... 4-101
ExtINT Interrupt and Task Priority ... 4-101
Removing Masks ... 4-101
Delivery Mode and Trigger Mode ... 4-101
Assigning Interrupt Vectors ... 4-102
Sending Inter-Processor Interrupts ... 4-102
Delay with Level Triggered Interrupts ... 4-102
Reset Deassert ... 4-102
Interrupt Masking ... 4-102
Changing Redirection Tables ... 4-102
Device Drivers with 82489DX ... 4-102

SYSTEM HARDWARE AND SOFTWARE DESIGN CONSIDERATIONS ... 4-103

DIRECTIONS FOR EASY MIGRATION TO FUTURE INTEGRATED APIC ... 4-106

1.0 INTRODUCTION

The 82489DX Advanced Programmable Interrupt Controller provides multiprocessor interrupt management, providing both static and dynamic symmetrical Interrupt distribution across all processors.

The main function of the 82489DX is to provide interrupt management across all processors. This dynamic interrupt distribution includes routing of the interrupt to the lowest-priority processor. The 82489DX works in systems with multiple I/O subsystems, where each subsystem can have its own set of interrupts. This chip also provides inter-processor interrupts, allowing any processor to interrupt any processor or set of processors. Each 82489DX I/O unit Interrupt Input pin is individually programmable by software as either edge or level triggered. The interrupt vector and interrupt steering information can be specified per pin. A 32-bit wide timer is provided that can be programmed to interrupt the local processor. The timer can be used as a counter to provide a time base to software running on the processor, or to generate time slice interrupts locally to that processor. The 82489DX provides 32-bit software access to its internal registers. Since no 82489DX register reads have any side effects, the 82489DX registers can be aliased to a user read-only page for fast user access (e.g., performance monitoring timers).

The 82489DX supports a generalized naming/addressing scheme that can be tailored by software to fit a variety of system architectures and usage models. It also supports 8259A compatibility by becoming virtually transparent with regard to an externally connected 8259A style controller, making the 8259A visible to software.

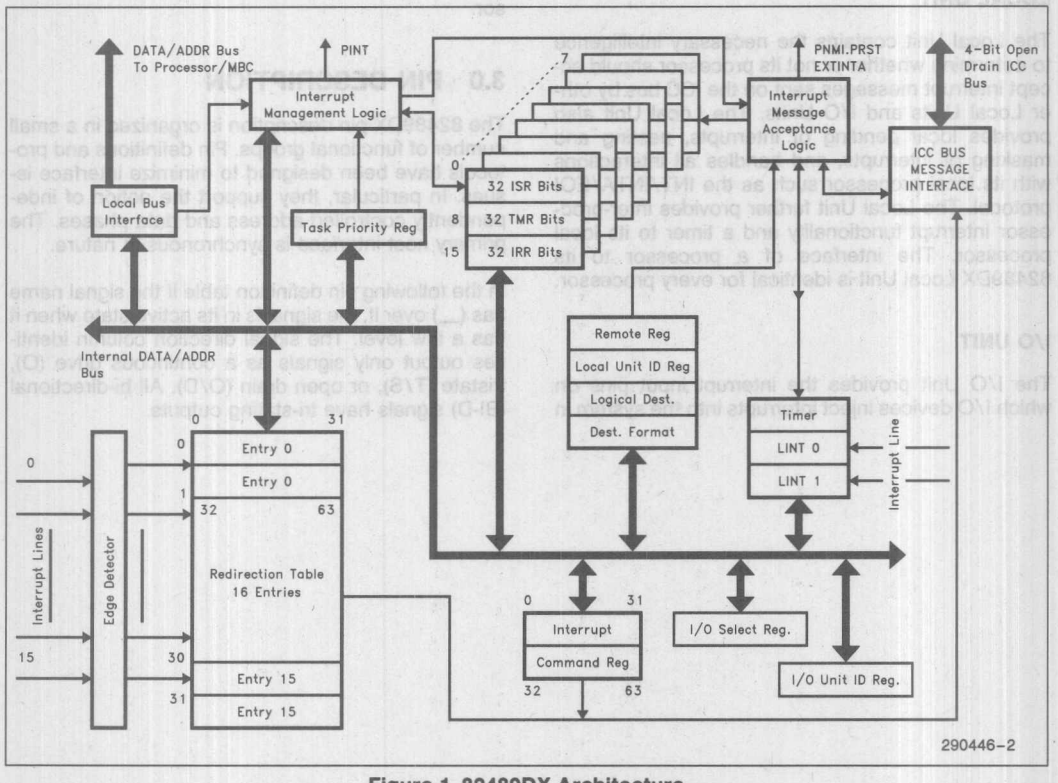


Figure 1. 82489DX Architecture

2.0 FUNCTIONAL OVERVIEW

82489DX Functional Blocks

82489DX contains one Local Unit, one I/O unit and a timer. The ICC bus is used to pass interrupt messages.

ICC BUS

The ICC bus is a 5-wire synchronous bus connecting all 82489DXs (all I/O Untis and all Local Units). The Local Units and I/O Units communicate over this ICC bus. Four of these five wires are used for data transmissions and arbitration, and one wire is a clock.

LOCAL UNIT

The Local Unit contains the necessary intelligence to determine whether or not its processor should accept interrupt messages sent on the ICC bus by other Local Units and I/O Units. The Local Unit also provides local pending of interrupts, nesting and masking of interrupts, and handles all interactions with its local processor such as the INT/INTA/EOI protocol. The Local Unit further provides inter-processor interrupt functionality and a timer to its local processor. The interface of a processor to its 82489DX Local Unit is identical for every processor.

I/O UNIT

The I/O Unit provides the interrupt input pins on which I/O devices inject interrupts into the system in the form of an edge or a level. The I/O unit also contains a Redirection Table for the interrupt input pins. Each entry in the Redirection Table can be individually programmed to indicate whether an interrupt on the pin is recognized as either an edge or a level; what vector and also what priority the interrupt has; and which of all possible processors should service the interrupt and how to select that processor (statically or dynamically). The information in the table is used to send interrupt messages to all 82489DX Units via the ICC bus.

TIMER

The 82489DX provides a 32-bit wide timer that can be programmed to interrupt the local processor. The timer can be used as a counter to provide a time-base to software running on the processor, or to generate time-slice interrupts local to that processor.

3.0 PIN DESCRIPTION

The 82489DX pin description is organized in a small number of functional groups. Pin definitions and protocols have been designed to minimize interface issues. In particular, they support the notion of independently controlled address and data phases. The primary host interface is synchronous in nature.

In the following pin definition table if the signal name has (__) over it, the signal is in its active state when it has a low level. The signal direction column identifies output only signals as a continuous drive (O), tristate (T/S), or open drain (O/D). All bi-directional (BI-D) signals have tri-stating outputs.

Pin Definition Table

Symbol	Pin No.	Type	Function
SYSTEM PINS			
RESET	65	I	The **RESET INPUT** forces 82489DX to enter its initial state. The 82489DX Local Unit in turn asserts it PRST (Processor Reset) output. All tri-state outputs remain in high impedance until explicitly enabled.
ExtINTA	41	O	The **EXTERNAL INTERRUPT ACKNOWLEDGE** output is asserted (high) when an external interrupt controller (e.g., 8259) is expected to respond to the current INTA cycle. If deasserted (low), 82489DX will respond, and the INTA cycle must not be delivered to the external controller.
CLKIN	57	I	**CLOCK INPUT** provides reference timing for most of the bus signals.
$\overline{\text{TRST}}$	56	I	**TEST RESET** is the JTAG compatible boundary scan TAP controller reset pin. A weak pull-up keeps the pin high if not driven.
TCK	55	I	**TEST CLOCK** is the clock input for the JTAG compatible boundary scan controller and latches.
TDI	53	I	**TEST DATA INPUT** is the test data input pin for the JTAG compatible boundary scan chain and TAP controller. A weak pull-up keeps this pin high if not driven.
TDO	52	O	**TEST DATA OUTPUT** is the test data output for the JTAG compatible boundary scan chain.
TMS	54	I	**TEST MODE SELECT** is the test mode select pin for the JTAG boundary scan TAP controller. A weak pull-up keeps this pin high if not driven.
TIMER PIN			
TMBASE	59	I	The **TIME BASE** input provides a standard frequency that is only used by the 82489DX timer and that is independent of the system clock.
INTERRUPT PINS			
INTIN[15:0]	82–97	I	These 16 **INTERRUPT INPUT** pins accept edge or level sensitive interrupt requests from I/O or other devices. The pin numbers are specified respectively. INTIN15 corresponds to pin number 82, INTIN14 corresponds to pin number 83 etc., and INTIN0 corresponds to pin number 97. These pins are active high.
LINTIN[1] LINTIN[0]	80 81	I I	Two **LOCAL INTERRUPT INPUT** pins accept edge or level sensitive interrupt requests that can only be delivered to the connected processor. These pins are active high.
REGISTER ACCESS PINS			
$\overline{\text{ADS}}$	64	I	**ADDRESS STROBE** signal indicating the start of a bus cycle. 82489DX does not commit to start the cycle internally until BUS GRANT is detected active.

Pin Definition Table (Continued)

Symbol	Pin No.	Type	Function
REGISTER ACCESS PINS (Continued)			
M/$\overline{IO}$, D/$\overline{C}$, W/$\overline{R}$	63 61 62	I I I	Bus cycle definition signals. Note that since the 82489DX registers can be mapped in either memory or I/O space, the M/$\overline{IO}$ pin is not used for register access cycles; it is only used to decode interrupt acknowledge cycles. 82489DX does not respond to code read cycles.
$\overline{BGT}$	66	I	The **BUS GRANT** input is optional and is used to indicate the address phase of a bus cycle in configurations where address timing cannot be inferred from $\overline{ADS}$. This signal is really used as an address latch enable, but is named as it is to indicate that it can normally be connected to the Intel Cache Controller generated signal of the same name. Must be tied low if not used.
$\overline{CS}$	74	I	The **CHIP SELECT** input indicates that the 82489DX registers are being addressed.
A3 A4 A5 A6 A7 A8 A9 A10	31 29 28 27 26 24 22 21	BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D	The address pins are used as inputs in addressing internal register space. Output function is reserved. They are also used to latch local unit ID on reset.
$\overline{DLE}$	73	I	**DATA LATCH/ENABLE** is optional and is used to indicate committing the data phase of a bus cycle in configurations where data timing cannot be inferred from other cycle timings. Must be tied low if not used.
D31 D30 D29 D28 D27 D26 D25 D24 D23 D22 D21 D20 D19 D18 D17 D16 D15 D14 D13 D12 D11	105 107 109 110 111 112 114 115 116 118 119 121 122 123 124 125 128 129 130 131 2	BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D	The DATA BUS is for all register accesses and interrupt vectoring.

Pin Definition Table (Continued)

Symbol	Pin No.	Type	Function
REGISTER ACCESS PINS (Continued)			
D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0	3 4 7 8 9 11 12 13 14 16 18	BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D BI-D	
DP3 DP2 DP1 DP0	101 102 103 104	BI-D BI-D BI-D BI-D	One Data Parity pin for each byte on the data bus. EVEN parity is generated any time the data bus is driven by the 82489DX.
$\overline{RDY}$	43	O	**READY** output indicates that the current bus cycle is complete. In the case of a read cycle, valid data and the return to inactive state after going active low may be delayed till $\overline{DLE}$ goes active.
PROCESSOR PINS			
PINT	35	T/S	The **PROCESSOR INTERRUPT OUTPUT** indicates to the processor that one or more maskable interrupts are pending. This pin is tri-stated at reset, and has an internal pull-down resistor to prevent false signaling to the processor until the 82489DX Local Unit is enabled and this pin is actively driven.
PRST	38	O	The **PROCESSOR RESET OUTPUT** is asserted/de-asserted upon 82489DX reset, and also in response to ICC bus messages with "RESET" delivery mode. This pin should be used with care.
PNMI	37	T/S	The **NON-MASKABLE INTERRUPT** output is signaled in respone to ICC bus messages with "NMI" delivery mode. This pin is tri-stated at reset, and has an internal pull-down resistor to prevent false signaling to the processor until the Local Unit is enabled and this pin is actively driven.
ICC BUS PINS			
ICLK	60	I	The **ICC BUS CLOCK** input provides synchronous operation of the ICC bus.
MBI[3:0]	76–79	I	The four **ICC BUS IN** inputs are used for incoming ICC bus messages. In smaller configurations the ICC bus input and outputs may be tied directly together at the pins. Pin number for MBI3 is 76, MBI2 is 77, MBI1 is 78 and MBI0 is 79.
MBO3 MBO2 MBO1 MBO0	45 48 49 51	O/D	The four **ICC BUS OUT** outputs are used for outgoing ICC bus messages. The current capacity is only 4 mA. So external bufferes will be needed.

Pin Definition Table (Continued)

Symbol	Pin No.	Type	Function
RESERVED PINS			
Reserved	34, 42	NC	These pins **MUST BE LEFT OPEN.**
Reserved	70, 72, 75		**Reserved** by Intel. **These pins should be strapped to V_{CC}.**
Reserved	71, 19, 20		**Reserved** by Intel. **These pins should be strapped to GND.**
POWER AND GROUND PINS			
V_{CC}	1, 32, 69, 98	POWER	Nominally +5V. These pins along with V_{SS} and V_{SSI} should be separately bypassed.
V_{CCP}	6, 15, 25,100, 108, 117, 126	POWER	Nominally +5V. These pins along with V_{SSP} should be separately bypassed.
V_{CCPO}	39, 46	POWER	Nominally +5V. These pins along with V_{SSPO} should be separately bypassed.
V_{SS}	5, 33, 67, 68, 99	GND	Nominally 0V. These pins along with V_{CC} should be separately bypassed.
V_{SSP}	10, 17, 23, 30, 106, 113, 120, 127, 132,	GND	Nominally 0V. These pins along with V_{CCP} should be separately bypassed.
V_{SSPO}	36, 40, 44, 47, 50	GND	Nominally 0V. These pins along with V_{CCPO} should be separately bypassed.
V_{SSI}	58	GND	Nominally 0V. These pins along with V_{CC} should be separately bypassed.

NOTE:
V_{CC}, V_{CCP} and V_{CCPO} should be of same voltage. V_{SS}, V_{SSP}, V_{SSPO} and V_{SSI} should be 0V.

4.0 FUNCTIONAL DESCRIPTION

As far as interrupt management is concerned, the 82489DX's interrupt control function spans over two functional units, the I/O Unit of which there is one per I/O subsystem, and the Local Unit of which there is one per processor. 82489DX has one I/O unit and one Local Unit in a single package. This section takes a detailed look at both local and I/O Units.

I/O Unit

The I/O Unit consists of a set of Interrupt Input pins, an Interrupt Redirection Table, and a message unit for sending and receiving messages from the ICC bus. The I/O Unit is where I/O devices inject their interrupts, the I/O Unit selects the corresponding entry in the Redirection Table and uses the information in that entry to format an interrupt request message. The message unit then broadcasts this message over the ICC bus. The content of the Redirection Table is under software control and is assigned benign defaults upon reset. The masks in the Redirectional Table entries are set to 1 at *hardware reset* to disable the interrupts.

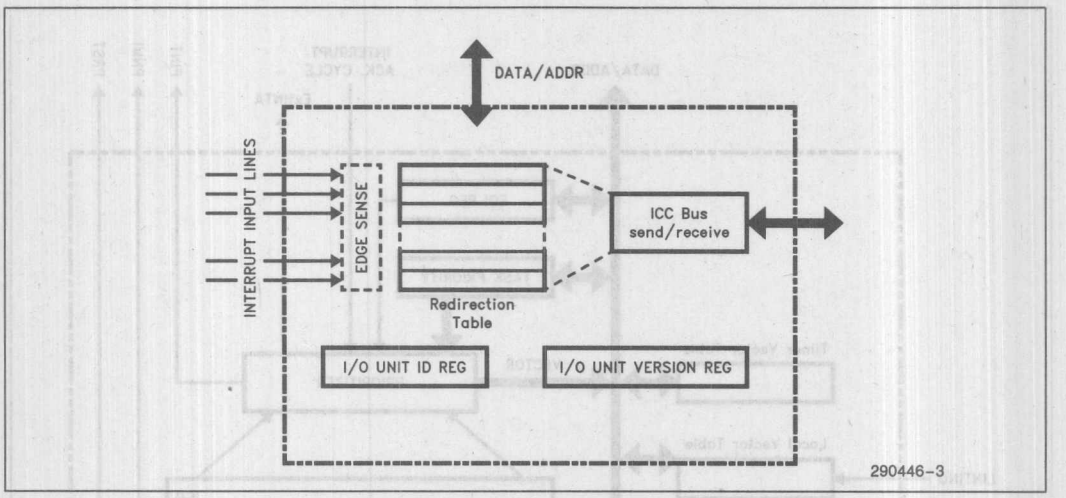


Figure 2. 82489DX I/O Unit Block Diagram

Local Unit

Interrupt Management of the Local Unit is responsible for local interrupt sources, interrupt acceptance, dispensing interrupts to the processor, and sending inter-processor interrupts. Depending on the delivery mode of the interrupt, zero, one or more units can accept an interrupt. A Local Unit accepts an interrupt only if it will deliver the interrupt to its processor. Accepting an interrupt is purely an inter-82489DX matter; dispensing an interrupt to the local processor only involves a 82489DX and its local processor.

4

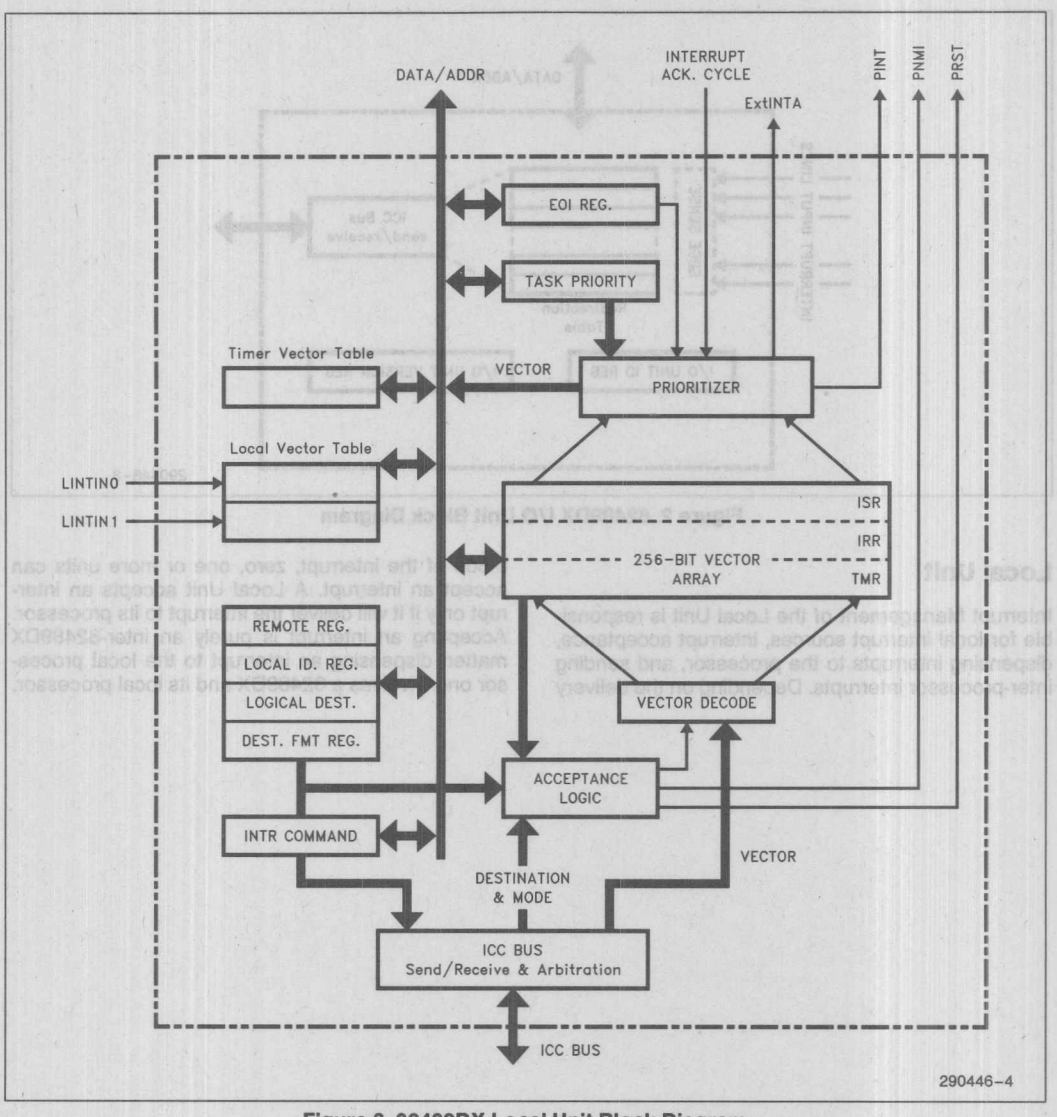


Figure 3. 82489DX Local Unit Block Diagram

5.0 INTERRUPT CONTROL MECHANISM

This section describes briefly the interrupt control mechanism in the 82489DX.

5.1 Interrupts

The interrupt control function of all 82489DXs are collectively responsible for delivering interrupts from interrupt sources to interrupt destinations in the multiprocessor system. When a processor accepts an interrupt, it uses the vector to locate the entry point of the handler in its interrupt table. The 82489DX architecture allows for 16 possible interrupt priorities; zero being the lowest priority and 15 being the highest. Priority of interrupt A "is higher than" the priority of interrupt B if servicing A is more urgent than servicing B. An interrupt's priority is implied by its vector; namely priority = vector/16.

With 256 vectors and 16 different priorities, this implies that 16 different interrupt vectors can share a single interrupt priority.

TOTAL ALLOWED INTERRUPT VECTORS

Out of 256 vectors, interrupt vectors 0 to 15 should not be used in the 82489DX. Only 240 interrupt vectors (vectors from 16 to 255) are supported in the 82489DX.

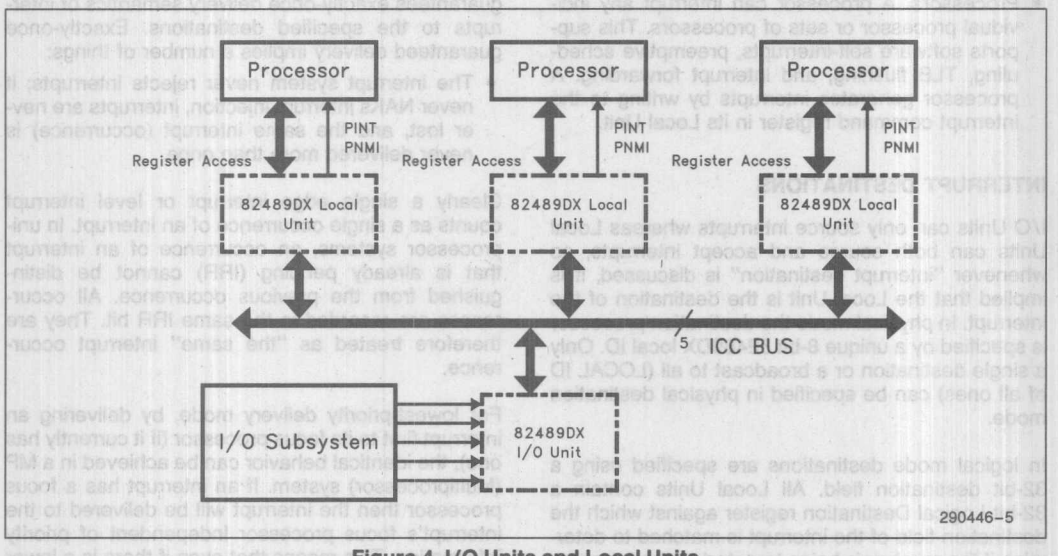


Figure 4. I/O Units and Local Units

4

INTERRUPT SOURCES

Interrupts are generated by a number of different interrupt sources in the system.

Possible interrupt sources are:

- Externally connected (I/O) devices. Interrupts from these external sources manifest themselves as edges or levels on interrupt input pins and can be redirected to any processor.
- Locally connected devices. These originate as edges or levels on interrupt pins, but they are al ways directed to the local processor only.
- 82489DX timer generated interrupts. Like locally connected devices, 82489DX timer can only interrupt its local processor.
- Processors. A processor can interrupt any individual processor or sets of processors. This supports software self-interrupts, preemptive scheduling, TLB flushing, and interrupt forwarding. A processor generates interrupts by writing to the interrupt command register in its Local Unit.

INTERRUPT DESTINATIONS

I/O Units can only source interrupts whereas Local Units can both source and accept interrupts, so whenever "interrupt destination" is discussed, it is implied that the Local Unit is the destination of the interrupt. In physical mode the destination processor is specified by a unique 8-bit 82489DX local ID. Only a single destination or a broadcast to all (LOCAL ID of all ones) can be specified in physical destination mode.

In logical mode destinations are specified using a 32-bit destination field. All Local Units contain a 32-bit Logical Destination register against which the destination field of the interrupt is matched to determine if the receiver is being targeted by the interrupt. An additional 32-bit Destination Format register in each Local Unit enables the logical mode addressing.

INTERRUPT DELIVERY

The description of interrupt delivery makes frequent use of the following terms:

- Each processor has a processor priority that reflects the relative importance of the code the processor is currently executing. This code can be part of a process or thread, or can be an interrupt handler. A processor's priority fluctuates as a processor switches threads, a thread or handler raises and lowers its priority level to mask out interrupt, and the processor enters an interrupt handler and returns from an interrupt handler to previously interrupted activity.
- A processor is lowest priority within a given group of processors if its processor priority is the lowest of all processors in the group. Note that more than one processor can be the lowest priority in a given group.
- A processor is the focus of an interrupt if it is currently servicing that interrupt, or if it currently has a request pending for the interrupt.

Interrupt delivery begins with an interrupt source injecting its interrupt into the interrupt system at one of the 82489DX. Delivery is complete only when the servicing processor tells its 82489DX Local Unit it is complete by issuing an end-of-interrupt (EOI) command to its 82489DX Local Unit. Only then has all (relevant) internal state regarding that occurrence of the interrupt been erased. The interrupt system guarantees exactly-once delivery semantics of interrupts to the specified destinations. Exactly-once guaranteed delivery implies a number of things:

- The interrupt system never rejects interrupts; it never NAKs interrupt injection, interrupts are never lost, and the same interrupt (occurrence) is never delivered more than once.

Clearly a single edge interrupt or level interrupt counts as a single occurrence of an interrupt. In uniprocessor systems, an occurrence of an interrupt that is already pending (IRR) cannot be distinguished from the previous occurrence. All occurrences are recorded in the same IRR bit. They are therefore treated as "the same" interrupt occurrence.

For lowest-priority delivery mode, by delivering an interrupt first to its focus processor (if it currently has one), the identical behavior can be achieved in a MP (Multiprocessor) system. If an interrupt has a focus processor then the interrupt will be delivered to the interrupt's focus processor independent of priority information. This means that even if there is a lower priority processor compared to the focus processor, the interrupt still gets delivered to the focus processor.

Each edge occurring on an edge triggered interrupt input pin is clearly a one-shot event; each occurrence of an edge is delivered. An active level on a level triggered interrupt input pin represents more of a "continuous event". Repeatedly broadcasting an interrupt message while the level is active would cause flooding of the ICC bus, and in effect transmits very little useful information since the same processor (the focus) would have to be the target.

Instead, for level triggered interrupts the 82489DX merely recreate the state of the interrupt input pin at the destination . The source 82489DX accomplishes this by tracking the state of the appropriate destina-

tion 82489DX's Interrupt Request Register (or pending bit) and only sending inter-82489DX messages when the state of the interrupt input pin and the destination's interrupt request enter a disagreement. Unlike edge triggered interrupts, when a level interrupt goes into service, the interrupt request at the servicing 82489DX is not automatically removed. If the handler of a level sensitive interrupt executes an EOI then that interrupt will immediately be raised to the processor again, unless the processor has explicitly raised its task priority, or the source of that interrupt has been removed.

5.2 Interrupt Redirection

This section specifically talks about how a processor is picked during interrupt delivery. The 82489DX supports two modes for selecting the destination processor: Fixed and Lowest Priority.

- *Fixed Delivery Mode*
  In fixed delivery mode, the interrupt is unconditionally delivered to all local 82489DXs that match in the destination information supplied with the interrupt. Note that for I/O device interrupts typically only a single 82489DX would be listed in the destination. Priority and focus information are ignored. If the priority of a destination processor equal to or higher than the priority of the interrupt, then the interrupt is held pending locally in the destination processor's Local Unit, until the processor priority becomes low enough at which time the interrupt is dispensed to the processor. More than one processor can be the destination in fixed-delivery mode.
- *Lowest Priority Delivery Mode*
  Under the lowest priority delivery mode, the processor to handle the interrupt is the one in the specified destination with the lowest processor priority value. If more than one processor is at the lowest priority, then a unique arbitration ID is used to break ties. For lowest priority dynamic delivery, the interrupt will always be taken by its focus processor if it has one. The lowest priority delivery method assures minimum interruption of high priority tasks. Since each Local Unit only knows its own processor priority, determining the lowest priority processor is done by arbitration on the ICC bus. Only one processor can be the destination in lowest-priority delivery mode.

INTER-82489DX COMMUNICATION

All I/O and Local Units communicate during interrupt delivery. Interrupt information is exchanged between different units on a dedicated five wire ICC bus in the form of broadcast messages. A 82489DX Unit's 8-bit ID is used as its name for the purpose of using the ICC bus, and all 82489DX units using one ICC bus should be assigned a different ID. The Arbitration ID of the Local Units used to resolve ties during lowest priority arbitration is also derived from the Local Unit's ID.

6.0 82489DX LOCAL UNIT REGISTERS DESCRIPTION

6.1 Local Unit ID Register

Each 82489DX Local Unit has a register that contains the Local Unit's 8-bit ID. The Local Unit ID serves as a physical name of the 82489DX Local Unit. It can be used in specifying destination information and is also used for accessing the ICC bus. Eight address lines A[10]–A[3] are sampled on every clock edge while RESET is asserted. The last sample remains in the Local Unit ID register after reset. Alternatively, the ID can be loaded with a register write as part of software initialization. The Local Unit ID is read-write by software.

Bits [31..24]	Bits [23..0]

82489DX Local Unit ID Register

Bits [31..24] Local Unit ID: The Local Unit ID serves as the physical "name" of the Local Unit used for addressing the 82489DX in physical destination mode and for the ICC bus usage. In a system with say four 82489DX, there are 4 Local Units and 4 I/O Units. All the 8 units should be assigned different IDs. For future compatibility use only IDs from 0 to 14.

Bits [23..0] Bits [23..0] are Reserved. They should be written with 0.

6.2 Destination Format Register

Interrupt Destination can be either addressed physically or logically. When the interrupt message addresses the destination physically, each 82489DX in the ICC bus compares the address with its own unit ID. If the message is a broadcast type then every Local Unit accepts the interrupt.

When the message addresses the destination using logical addressing scheme each Local Unit in the ICC bus compares the logical address in the interrupt message with its own Logical Destination Register. If there is a bit match (i.e., if at least one of the corresponding pair of bits match) this local unit is selected for delivery.

82489DX connected in the ICC bus should be written with "1" to enable the addressing scheme.

Destination Format Register

Bits [31:0]

Logical Destination Register

Bits [31:0]

For future compatibility, use only bits 31–24 of Logical Destination Register. For binary compatibility, it is strongly recommended that 82489DX software use only 8 MSB of Logical Destination Register.

6.3 Local Interrupt Vector Table Registers

The Redirection Table serves to steer interrupts that originate in the I/O subsystems to the processors. The Local Vector Table is its equivalent for interrupts that are restricted to only the local processor. The Local Vector Table contains three 32-bit registers. Register 0 corresponds to the timer, registers 1 and 2 correspond to local interrupt input pins, LINTIN0 and LINTIN1.

The format of both the Local 0 and Local 1 interrupt vector tables are identical. The following register description talks about both Local Interrupts 0,1 vectors.

Local Interrupts 0, 1 Interrupt Vectors

Vector: [Bits 7–0]

This is the vector to use when generating an interrupt for this entry.

Delivery Mode: [Bits 10–8]

000: Fixed

001: <reserved>

010: <reserved>

011: <reserved>

100: NMI

101: <reserved>

111: ExtINT

000: (fixed) means deliver the signal on the INT pin of the local processor. Trigger mode for "fixed" Delivery Mode can be edge or level.

100: (NMI) means deliver the signal on the NMI pin of the local processor. Vector information is ignored. A Delivery Mode equal to "NMI" requires a "level" triggered mode.

111: (ExtINTA) means deliver the signal to the INT pin of the local processor as an interrupt that originated in an externally connected (8259A compatible) interrupt controller. ExtINTA pin is as- serted also. The INTA cycle that corresponds to this ExtINTA delivery, should be routed to the external controller that is expected to supply the vector. A delivery mode of ExtINT requires an edge trigger mode. (See the section on compatiblity for more details.)

Bit 11: Bit 11 is Reserved. It should be written 0.

Delivery Status: [Bit 12]

This field is software-read only. Software writes to this field (as part of a 32-bit word) have no effect on this bit. Delivery status is a 1-bit field that contains the current status of the delivery of this interrupt. Two states are defined.

0: (idle) means that there is currently no activity for this interrupt.

1: (send pending) indicates that the interrupt has been injected, but its delivery is temporarily held up by the recently injected interrupts that are in the process of being delivered.

Local INT0 Vector Table

Bits [31:17]	Bit 16	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bits [10:8]	Bits [7:0]

Local INT1 Vector Table

Bits [31:17]	Bit 16	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bits [10:8]	Bits [7:0]

Figure 5. Local Vector Table

Bit 13: Bit 13 is Reserved. It should be written 0.

Remote IRR: [Bit 14]

This bit is used for level triggered local interrupts; its meaning is undefined for edge triggered interrupts. Remote IRR mirrors the interrupt's IRR bit of this local unit. Remote IRR is software read-only; software writes to this bit do not affect it.

Trigger Mode: [Bit 15]

The Trigger Mode field indicates the type of signal on the local interrupt pin that triggers an interrupt.

0: indicates edge sensitive,

1: indicates level sensitive.

Only the local interrupt pins can be programmed as edge or level triggered. Timer interrupts are always treated as edges.

MASK: [Bit 16]

0: enables injection of the interrupt,

1: masks injection of the interrupt.

Bits [31:17] Bits [31:17] are Reserved. Should be written 0.

6.4 Inter-Processor Interrupt Registers

A processor generates inter-processor interrupts by writing to the Interrupt Command Register in its 82489DX Local Unit. Conceptually, this can be thought of as the processor providing the interrupt's Redirection Table Entry on the fly. Not surprisingly, the layout of the Interrupt Command Register resembles that of an entry in the Redirection Table. Note that the format of this register allows a processor to generate any interrupt. A processor may use this to forward device interrupts originally accepted by it to other processors.

All fields of the Interrupt Command Register are read-write by software with the exception of the Delivery Status field which is read-only. Writing to the 32-bit word that contains the interrupt vector causes the interrupt message to be sent.

Vector: [Bits 7–0]

The vector identifies the interrupt being sent. If the Delivery Mode is "Remote Read", then the Vector field contains the address of the register to be read in the remote 82489DX's Local Unit. The addresses are listed in the section discussing 82489DX Local Unit register summary. For example, for ID register, remote read address of 02 should be specified in vector field.

Delivery Mode: [Bits 10–8]

The Delivery Mode is a 3-bit field that specifies how the 82489DX listed in the destination field (bits 63:32) should act upon reception of this signal. Note that certain Delivery Modes will only operate as intended when used in conjunction with a specific Trigger Mode. These restrictions are indicated for each Delivery Mode.

000: (Fixed) means deliver the signal on the INT pin of all processors listed in the destination. Trigger Mode for "fixed" Delivery Mode can be edge or level.

001: (Lowest Priority) means deliver the signal on the INT pin of the processor that is executing at the lower priority among all the processors listed in the specified destination; Trigger Mode for "lowest priority" Delivery Mode can be edge or level.

010: Intel Reserved. Should not be used.

011: (Remote Read) is a request to a remote 82489DX Local Unit to send the value of one of its registers over the ICC bus. The register is selected by providing its address in the Vector field. The register value is latched by the requesting 82489DX and stored in the Remote Register where it can be read by the local processor. A Delivery Mode of "Remote Read" requires an "edge" Trigger Mode.

Interrupt Command Register [31:0]

Bits [31:20]	Bits [19:18]	Bits [17:16]	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bits [10:8]	Bits [7:0]

100: (NMI) means deliver the signal on the NMI pin of all processors listed in the destination, vector information is ignored. A Delivery Mode equal to "NMI" requires a "level" Trigger Mode.

101: (Reset) means deliver the signal to all local units listed in the destination. The destination local unit will assert/deassert its PRST output pin. All addressed 82489DX Local Units will assume their reset state but preserve their ID. One side effect of an ICC bus message with Delivery Mode equal to "Reset" that results in a deassert of reset is that all Local Units (whether listed in the destination or not) will reset their lowest-priority tie breaker arbitration ID to their Local Unit ID (see the section on the ICC bus for details). A Delivery Mode of "Reset" requires a "level" Trigger Mode. "RESET" should not be used with "self" or "all incl self" Shorthand mode since it will leave the system in non-recoverable reset state. If "RESET" is used with "all excl self" mode software should make sure that only one CPU executes this instruction in a MP system.

110: Intel Reserved. Should not be used.

111: Intel Reserved. Should not be used

Destination Mode: [Bit 11]

This field determines the interpretation of the Destination field.

0: (Physical Mode): in Physical Mode, a destination 82489DX is identified by its Local Unit ID. Bits 56 through 63 (8 MSB of the destination field) specify the 8-bit 82489DX Local Unit ID.

1: (Logical Mode): in Logical Mode, destinations are identified by matching on Logical Destination under the control of the Destination Format Register in each Local 82489DX. The 32-bit Destination field is the logical destination.

Delivery Status: [Bit 12]

Delivery Status is a 1-bit field that contains the current status of the delivery of this interrupt. Two states are defined:

0: (Idle) means that there is currently no activity for this interrupt;

1: (Send Pending) indicates that the interrupt has been injected, but its delivery is temporarily held up by other recently injected interrupts that are in the process of being delivered;

Delivery Status is software read-only; software writes to this field (as part of a 32-bit word) do not affect this bit. Software can read to find out if the current interrupt has been sent, and the Interrupt Command Register is available to send the next interrupt. If the Interrupt Command Register is overwritten before the Delivery Status is "Idle", then the destiny of that interrupt is undefined; i.e., the interrupt may have been lost.

Bit 13: Bit 13 is Reserved . Should be written 0.

Level: [Bit 14]

Software can use this bit in conjunction with the Trigger Mode bit when issuing an inter-processor interrupt to simulate assertion/deassertion of level sensitive interrupts.

To assert: Trigger mode = 1 and Level = 1.

To deassert: Trigger mode = 1 and Level = 0.

For example, a message with Delivery Mode of "Reset", a Trigger Mode of "Level", and Level bit of 0 deasserts Reset to the processor of the addressed 82489 DX Local Unit(s). As a side effect, this will also cause all 82489DX to reset their Arbitration ID to their unit ID. (The Arb ID is used for tie breaking in lowest priority arbitration.)

Trigger Mode: [Bit 15]

Software can use this in conjunction with Level Assert/Deassert to generate interrupts that behave as edges or levels.

0: Edge

1: Level

Remote Read Status: [Bits 17,16]

This field indicates the status of the data contained in the Remote Read register. This field is read-only to software. Whenever software writes to the Interrupt Command Register using Delivery Mode "Remote Read" the Remote Read status becomes "in-progress" (waiting for the remote data to arrive). The remote 82489DX Local Unit is expected to respond in a fixed amount of ICC bus cycles. If the remote 82489DX Local Unit is unable to do so, then the Remote Read status becomes "Invalid". If successful, the Remote Read status resolves to "Valid". Software should poll this field to determine completion and success of the Remote Read command.

00: (invalid): the content of the Remote Read Register is invalid. This is the case after a Remote Read command issued and the remote 82489DX Local Unit was unable to deliver the Register content in time.

01: (in progress): a Remote Read command has been issued and this 82489DX is waiting for the data to arrive from the remote 82489DX Local Unit.

10: (valid): the most recent Remote Read command has completed and the remote register content in the Remote Read Register is valid.

11: reserved.

Destination Shorthand: [Bits 19,18]

This field indicates whether a shorthand notation is used to specify the destination of the interrupt and if so, which shorthand is used. Destination Shorthands do no use the 32-bit Destination field, and can be sent by software with a single 32-bit write to the 82489DX's Interrupt Command Register. Shorthands are defined for the following common cases: software self interrupt, interrupt to all processors in the system including the sender, interrupts to all processors in the system excluding the sender.

00: (dest field): means that no shorthand is used. The destination is specified in the 32-bit Destination field in the second word (bits 32 to 63) of the Interrupt Command Register.

01: (self): means that the current 82489DX Local Unit is the single destination of the interrupt. This is useful for software interrupts. The Destination field in the Interrupt Command Register is ignored. RESET Delivery mode should not be used with self destination. Only FIXED delivery mode should be used with SELF.

10: (all incl self): means that the interrupt is to be sent to "all" processors in the system including the processor sending the interrupt. The 82489DX will broadcast a message with destination unit ID field set to all ones. RESET assert Delivery mode should not be used with "all incl self" destination.

11: (all excl self): means that the interrupt is to be sent to "all" processors in the system with the exception of the processor sending the interrupt. The 82489DX will broadcast a message with destination unit ID field set to all ones. All-excl-self is useful during selection of a boot processor (init) and also for TLB flush where "self" is flushed using the processor flush instruction. Only one CPU in a MP system should execute "all excl self" destination if used with RESET Delivery mode.

Bits [31:20] Bits [31:20] are Reserved. They should be written 0.

Interrupt Command Register [63:32]

Bits [63:32]

Destination: [Bits 63–32]

This field is only used when the Destination Shorthand is set to "Dest Field". If Destination Mode is Physical Mode, **then the 8 MSB contain a Destination unit ID.** If Logical Mode, the full 32-bit Destination field contains the logical address. The enabling is done by Destination Format Register.

6.5 IRR, ISR, TMR Registers

INTERRUPT ACCEPTANCE

All 82489DX Local Units listen to all messages sent over the ICC bus. For each message, the local unit first checks if it belongs to the destination in the

message. It does this by matching the 32-bit Destination field in the message against its logical Destination Register, if the message addresses in logical mode, and against its physical ID, if the message addresses in physical mode. All 82489DX Local Units that match are said to "belong to the group".

Each 82489DX Local Unit contains three 256-bit registers that play a role in the acceptance of interrupts and in dispensing accepted interrupts to the local processor. Each of these registers is a bit array where bit position i tracks information about the interrupt with vector i. These bits track information about the (PINT) maskable interrupts only. They are not relevant for NMI, RESET or ExtINT type of interrupts. The Interrupt Request Register (IRR) contains the interrupts accepted by this 82489DX Local Unit but not yet dispensed to the processor. The In Service Register (ISR) contains the interrupts that are currently in service by the processor, i.e., the interrupts that have been dispensed to the processor but for which the processor has not yet signaled the End-Of-Interrupt.

Note that the 82489DX's IRR and ISR registers have the same meaning and operation as in the 8259A in fully nested/non-specific EOI mode. Note also that these registers play no role in providing 8259A compatibility. Compatibility is handled by making an external 8259A-style controller directly visible to the processor and having the 82489DX become transparent.

Each interrupt has a vector associated with it, which determines the bit position, and hence the priority for the interrupt. When an interrupt is being serviced, all equal or lower priority interrupts are automatically masked by the 82489DX Local Unit.

The Trigger Mode Register (TMR) indicates for each interrupt whether the interrupt is edge or level. This information is transmitted with each 82489DX interrupt request message and reflects the Trigger Mode bit in the interrupt's Redirection Table entry. If an interrupt goes in service and the TMR bit is 0 (edge), then the interrupt's IRR bit is cleared at the same time the ISR bit is set. If the TMR bit is 1 (level), then the IRR bit is not cleared when the interrupt goes in service. In the latter case, the IRR bit mirrors the state of the interrupt's input pin.

The following diagram shows 82489DX operation with devices A and B sharing a level triggered interrupt input. The diagram illustrates how Remote IRR, and the IRR bit at the destination 82489DX track the state of INTIN. It also illustrates how an EOI is followed immediately be re-raising the interrupt as long as the INTIN is still asserted by some device.

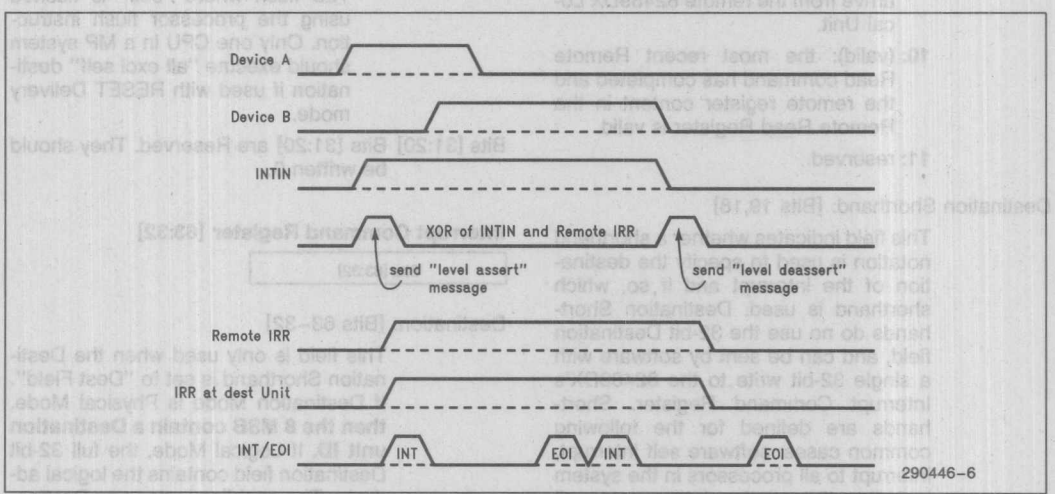


Figure 6. Interrupt Sharing

ISR, IRR, and TMR are read-only by software. Each of these 256-bit registers is accessed as four separate 32-bit registers. Note that there is no general Interrupt Mask Register (IMR) as in the 8259A. The processor masks interrupts temporarily by writing to the lcoal unit's Task Priority Register (described shortly).

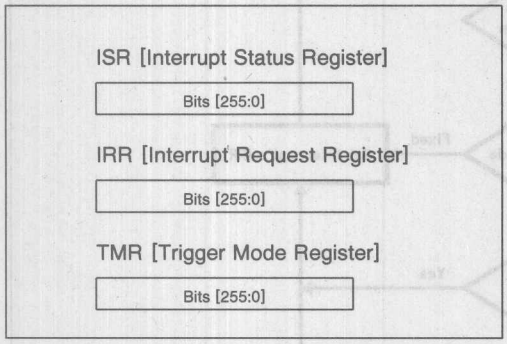


Figure 7. ISR, IRR, and TMR

TMR (Trigger Mode Register):

If 0 [edge triggered] the corresponding IRR bit is automatically cleared when interrupt service starts. If 1[level triggered] this is not the case; instead, the source 82489DX must explicitly request the IRR bit be cleared (upon deassert of the interrupt input pin or upon sending an appropriate interprocessor interrupt). Upon acceptance of an interrupt, the TMR bit is cleared for edge triggered interrupts and set for level triggered interrupts. This information is carried in the accepted interrupt message. The source 82489DX I/O unit also tracks the state of the destination unit's IRR bit (Remote IRR bit in the Redirection Table). When a level triggered interrupt input is deasserted, the source 82489DX I/O unit detects the discrepancy between the input pin state and the Remote IRR, and automatically sends a message telling the destination 82489DX to clear IRR for the interrupt.

IRR (Interrupt Request Register):

It contains the active interrupt requests that have been accepted, but not yet dispensed by this 82489DX Local Unit. A bit in IRR is set when the 82489DX Local Unit accepts the interrupt. When TMR is 0, it is cleared when the interrupt is serviced; when TMR is 1, it is cleared when the 82489DX Local Unit receives a message to clear it.

ISR (In Service Register):

It marks the interrupts that have been delivered to the processor, but that have not been fully serviced in that an End-Of-Interrupt has not yet been received. The ISR register reflects the current state of the processor's interrupt stack.

ACCEPTANCE MECHANISM

Interrupt acceptance proceeds as follows. If the delivery mode is Fixed, then each unit in the destination group unconditionally accepts the interrupt message and sets the interrupt's IRR bit. If the delivery mode is Lowest Priority, then each processor in the group first checks if it is currently the focus of the interrupt by checking its ISR and IRR. If an 82489DX finds one of these bits set for the incoming interrupt, then that 82489DX Local Unit accepts the interrupt independent of priority, and "signals" the other 82489DX Local Units to abort the priority arbitration. This avoids multiple delivery of a same interrupt occurrence to different processors, consistent with interrupt delivery semantics in uniprocessor systems as described above. If a message is to be delivered for NMI or Reset, then all 82489DX Local Units listed in the destination unconditionally assert/deassert the corresponding output pin. ISR, IRR, etc. are bypassed for NMI or reset and vector information is undefined.

The acceptance decision process is illustrated in the flow chart below.

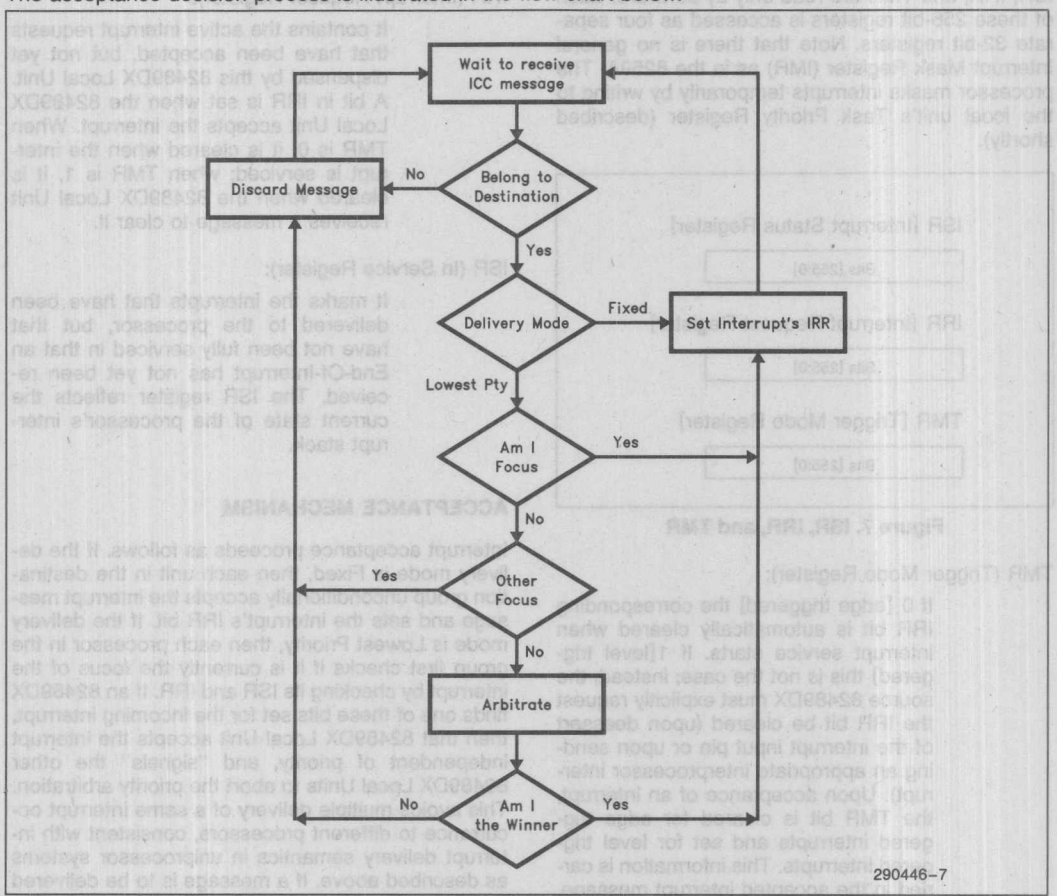


Figure 8. Interrupt Acceptance Flow Chart

6.6 Tracking Processor Priority

Each 82489DX Local Unit should be programmed with task priority so that it can mask interrupts that are less priority than that of the processor temporarily.

Task switching and task priority changes are the result of explicit software action. The operating system may define a number of task scheduling classes. Examples are an idle class, a background class, a foreground class, and a time critical class. Alternatively, different classes can be assigned to user code versus system code. If tasks in different classes are executing when an interrupt comes in, then it may be advantageous to interrupt the processor currently running the task in the least important class. Clearly, if one processor is idle while others are doing work, the idle processor would be the obvious target for servicing the interrupt. This implies that there is use in defining priority levels below all interrupt levels that can participate in lowest priority delivery selection.

At times, the operating system may need to block out interrupts from being serviced. For example, to synchronize access to a shared data structure between a device driver and its interrupt handler the driver raises it priority to equal or higher than the interrupt's priority.

The local 82489DX supports this via its Task Priority Register (the 8259A supports this via the interrupt mask register (IMR).) Software that wants to make use of this is required to inform its 82489DX Local Unit of the prioity change by updating the Task Priority Register. The Task Priority field is 8 bits providing up to 256 distinct priorities. The 4 MSB of this register correspond to the 16 interrupt priorities while the 4 LSB provide more precision. Priorities are best noted as x:y, where x is the value of the 4 MSB and y is the value of the 4 LSB. For example, Task Priority Register values 0:y with $0 < y < 15$ (and 0 in the 4 MSB) can be used to represent the priorities of the task scheduling classes described above ($y = 0$ for idle; $y = 1$ for background; etc.). Except for interrupts with vectors 0 through 15 (which are often predefined by the processor) which all have priority 0:0, the priorities of all other interrupts and their handlers is x:0 with $1 < x < 15$ and is above the base task priorities 0:y.

For example, interrupt vector 123 has priority 7:0 ($123/16 = 7$) and can be masked by any task that raises its priority to a value equal or higher than 7:0.

82489DX uses Task priority register for the purpose of masking the interrupts. The task priority register should be programmed with a priority value to specifiy the priority of task the processor is executing. 82489DX masks any interrupts of lower or equal priority when compared with task priority.

When task priority register is programmed with the priority 15, all the interrupts are masked. When task priority register is programmed with priority level X, by definition, all the interrupts of priority X and below X will be masked. When task priority register is programmed with the priority 0 then all the interrupts above priority 0 are allowed to interrupt the processor. This means that when task priority register is programmed even with the lowest value, i.e., 0, interrupts of priority 0 will be masked. So only 240 interrupt vectors should be used in 82489DX. Interrupt vecotrs from 0 to 15 should not be used.

The first priority value computed is the maximum of:

- Task Priority (4msb : 4lsb) and
- the priority of the highest order ISR bit set ((vector/16) :0).

The value is used to determine whether or not a pending interrupt can be dispensed to the processor.

The second priority value computed is the maximum of:

- Task Priority (4msb : 4lsb), and
- the priority of the highest order ISR bit set ((vector/16) :0), and
- the priority of the highest order IRR bit set ((vector/16) :0).

This value is used during arbitration as part of lowest-priority delivery.

Task Priority Register

Bits [31:8]	Bits [7:0]

From the information in the Task Priority Register and the priority information derived from the ISR and IRR register, the 82489DX Local Unit computes two additional priority values:

Bits [31:8] Bits [31:8] are Reserved. They should be written 0.

Bits [7:0] Task Priority

Bits [7:0] are used to specify the task priority.

6.7 Dispensing Interrupts

DISPENSING INTERRUPTS TO THE LOCAL PROCESSOR

Once a 82489DX Local Unit accepts an interrupt, it guarantees delivery of the interrupt to its local processor. (This part of the 82489DX functions similarly to an 8259A.) Dispensing a maskable interrupt to the local processor begins when the Local Unit asserts the INT pin of its processor. If the processor has interrupts enabled, it will respond by issuing an INTA cycle. This causes the Local Unit to freeze its internal priority state and release the 8-bit vector of the highest priority interrupt on the data bus where it is read by the processor and used to find the handler's entry point. The INT/INTA protocol also causes the interrupt's ISR bit to be set. The corresponding bit in the IRR register is only cleared if the TMR register indicates it should do so (edge triggered interrupts), otherwise (level triggered interrupts), IRR is only cleared when the Interrupt Input Pin is deasserted.

6.8 Spurious Interrupt Vector Register

SPURIOUS INTERRUPT

Note that it can happen that a level-triggered interrupt is deasserted right before its INTA cycle. In that case, all IRR bits may be clear and the prioritizer may not find a vector to give to the processor. To satisfy the processor's demand for a vector, instead, the 82489DX will return a spurious interrupt vector instead.

A similar situation may occur when the processor raises its Task Priority at or above the level of the interrupt for which the Processor INT pin is currently being asserted. When the INTA cycle is issued, the interrupt that was to be dispensed has become masked (masked but remembered).

Dispensing the spurious interrupt vector does not affect the ISR register, so the handler for this vector should just return without EOI. If the vector is shared with a valid interrupt, then the handler can read the vector's bit in the ISR register to check if it is invoked for the valid interrupt (ISR bit set) or not (ISR bit clear). Given the range of 240 vectors, overloading the spurious interrupt with a valid interrupt is not expected to be common practice. The spurious interrupt vector to be used by a Local Unit is programmable via the Spurious Interrupt Vector Register.

UNIT ENABLE

It is possible that Local Units exist in the system that do not have a processor to which to dispense interrupts. The only danger this represents in the system is that if any interrupt is broadcast to all processors using lowest priority delivery mode when all processors are at the lowest priority, there is a chance that a Local Unit without the processor may accept the interrupt if this Local Unit happens to have the lowest Arb ID at the time. To prevent this from happening, all Local Units initialize in the disabled state and must be explicitly enabled before they can either start accepting or transmitting messages from the ICC bus. A disabled 82489DX Local Unit only responds to messages with Delivery Modes set to "Reset". Reset deassert messages should be sent in Physical Destination mode using the target's Local Unit ID since the logical destination information in the lcoal units is undefined (all zeroes) when the 82489DX comes out of Reset.

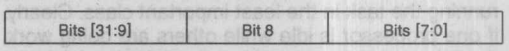


Figure 9. Spurious Interrupt Vector Register

Bits [31 .. 9] Reserved Bits. Should be written 0.

Unit Enable:[Bit 8]

0: When a 0 is written to this bit, this Local Unit gets disabled with regard to responding to messages sent as well as transmitting on the ICC bus. It only responds to messages with Delivery Mode set to "Reset". Reading a 0 at this bit indicates that the unit is disabled.

1: When a 1 is written to this bit, the current Local Unit is enabled for both transmitting and receiving unit messages. Reading a 1 at this bit indicates that the unit is enabled.

Spurious Vector: [Bits 7–0]

For future compatibility, bits [3–0] should be 1111.

6.9 End-of-Interrupt (EOI) Register

Before returning from the interrupt handler, software must issue an End-Of-Interrupt (EOI) command to the 82489DX Local Unit. The data written to EOI register is don't care. This tells the 82489DX to clear the highest priority bit in the ISR register since the interrupt is no longer in service. Upon EOI, 82489DX goes through prioritization returning to the next highest priority activity. This can be a previously interrupted handler (from ISR), a pending interrupt request (from IRR), or an interrupted task (from Task Priority).

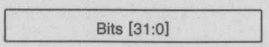


Figure 10. EOI Register

Bits [31:0]: are don't care.

6.10 Remote Read Register

Since all 82489DX Local Units would typically occupy the same address range, an 82489DX local unit's registers can only be accessed by the local processor. From a system debugging point of view, this would mean that a large amount of state would become inaccessible if its corresponding processor hangs for whatever reason. To assist in the debugging of MP systems, the 82489DX support a mechanism that provides read-only access to any register in any other 82489DX lcoal unit in the system.

To read any register in a "remote" 82489DX Local Unit, the processor writes to the Interrupt Command Register specifying a Delivery Mode equal to "Remote Read". The remote 82489DX is specified in the Destination field of the Interrupt Command Register in the usual fashion. Debug software would make sure that this selects a single 82489DX only—for example by using the target's 82489DX Local Unit ID in physical destination mode. Since no vector is associated with remote register access, the Vector field in the Interrupt Command Register is used to select the individual remote 32-bit register to be read. The selector value corresponds to the address (offset) of the register in the local 82489DX's address space. Sending a "Remote Read" command results in sending a message on the ICC bus. The destination 82489DX responds by placing the 32-bit content of the selected register on the ICC bus. This value is read by sending the 82489DX and place it in the Remote Register where software can get at it using regualr register access to its 82489DX Local Unit. The Remote Register is software read-only. The contents of the Remote Register is valid when the Delivery Status in the Interrupt Command Register has become "Idle" again.

Remote Read Register

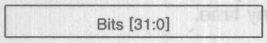


Figure 11. Remote Register

Bits [31:0] Bits [31:0] contain the contents of Remote Read Register.

6.11 82489DX Local Configuration

LOCAL VERSION REGISTER

Each 82489DX Local Unit contains a hardware Version Register that identifies this 82489DX Local Unit version. This register is read only.

Local Version Register

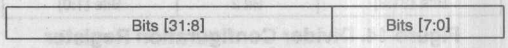


Figure 12. Local Version Register

Version: [Bits 7–0]

This is a version number that identifies this version. This field is hardwired and is read-only. Will be read as "1" for 82489DX.

Bits [31:8] Bits 31:8 are reserved.

6.12 82489DX Timer Registers

Overview

82489DX Local Unit contains one 32-bit wide programmable binary timer for use by the local processor. The timer can select its clock base from one of three possible clock inputs. A timer mode can be programmed to operate in either one-shot mode or periodic mode. The timer can be configured to interrupt the local processor with a vector.

Time Base

The 82489DX has two independent clock input pins:

1. The CLK pin provides the clock signal that drives the 82489DX's internal operation.
2. The TMBASE pin allows an independent clock signal to be connected to the 82489DX for use by the timer functions.

Signals from both CLK and TMBASE can be used as clock inputs that feed the timer. In addition, the 82489DX contains a divider that can be configured to divide either input clock signal. The divider can be programmed to divide the selected input clock by 2, 4, 8, or 16. CLK, TMBASE, and the output of the divider together provide three time bases: Base 0, Base 1, and Base 2. Base 0 is always equal to CLK: Base 1 is always equal to TMBASE; and base 2 is one of; CLK/2, CLK/4, CLK/8, CLK/16, TMBASE/2, TMBASE/4, TMBASE/8, or TMBASE/16. The timer can independently select one of these three time bases as its clock input as depicted in the following diagram.

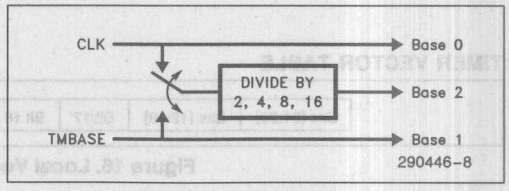


Figure 13. Time Bases

Bits [31:3]	Bit 2	Bits [1:0]

Figure 14. Divider Configuration Register

Bits [31:3] Bits 31 to 3 are reserved. They should be written 0.

Divider Input:

[Bit 2] Selects whether divider's input connects to the 82489DX Local Unit's CLK pin or TMBASE pin.

0: means the divider takes its input signal from CLK,

1: means use TMBASE.

Divide By: [Bits 1,0]

This field selects by how much the divider divides.

00: divide by 2

01: divide by 4

10: divide by 8

11: divide by 16

Timer

Software starts a timer going by programming its Initial Count Register. The timer copies this value into the Current Count Register and starts counting down at the rate of one count for each time base pulse. The time is one of Base 0, Base 1, or Base 2.

The timer has a programmable mode which can be One-Shot or Periodic. After the timer reaches zero in One-Shot mode, the timer simply stays at zero until it is reprogrammed. In Periodic mode, the timer automatically reloads its Current Count from the Initial Count and starts counting down again.

For the timer, interrupt generation can be disabled or enabled, and an arbitrary interrupt vector can be specified. When enabled and the timer reaches zero, an interrupt is generated at the 82489DX Local Unit. Timer generated interrupts are always treated as edges. They can only generate maskable interrupts to the local processor.

A timer set up with its interrupt masked is useful as a time base that can be sampled by the local processor by reading the Current Count Register, for the purpose of measuring the intervals. By mapping the 82489DX's register space into a read-only user page, safe and efficient performance monitoring of user programs can be supported.

If necessary, software may want to ensure that periodic timer interrupts on the different 82489DX Local Units are staggered such that the 82489DXs don't all deliver their interrupt (e.g., a timer slice interrupt) to their local processor at the same time. This staggering avoids bursts of contention for shared resources (bus, cache lines, dispatch queue, locks). Randomness occurring "naturally" may be sufficient to ensure staggering.

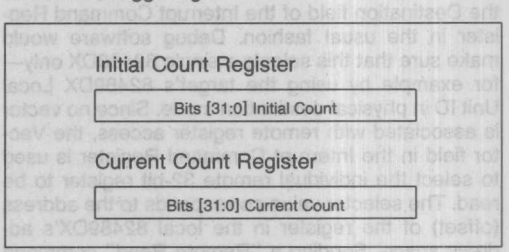


Figure 15. Initial Count and Current Count Registers

Initial Count: Software writes to this register to set the initial count for timer. This register can be written at any time. When written, its value is copied to the Current Count Register and countdown starts or continues from there. The Initial Count Register is read-write by software.

Current Count: This is the current count of timer. It is read-only by software and can be read at any time.

The timer is configured via its Local Vector Table entry shown below (see also Interrupt Control in this section).

Vector: [Bits 7–0]

This is the 8-bit interrupt vector to be used when timer generates an interrupt.

TIMER VECTOR TABLE

Bits [31:20]	Bits [19:18]	Bit 17	Bit 16	Bits [15:13]	Bit 12	Bits [11:8]	Bits [7:0]

Figure 16. Local Vector Table: Timer Entry

Bits 11–8 Reserved. Should be written 0.

Delivery Status: [Bit 12]

Delivery Status is a 1-bit field that contains the current status of the delivery of this interrupt. Two states are defined:

0: (Idle) means that there is currently no activity for this interrupt;

1: (Send Pending) indicates that the interrupt has been injected, but its delivery is temporarily held up by other recently injected interrupts that are in the process of being delivered; Delivery Status is software read-only; software writes to this field (as part of a 32-bit word) do not affect this bit.

Bits 15–13: Reserved. Should be written 0.

MASK: [Bit 16]

This bit serves to mask timer interrupt generation.

0: means not masked, when timer reaches 0, it generates an interrupt with vector at the 82489DX Local Unit

1: means masked, and no interrupt is generated.

Timer Mode: [Bits 17]

This field indicates the operation mode of timer.

0: (One-Shot): the Current Count Register remains at zero after the timer reaches zero, and software needs to reassign the timer's Initial Count Register to rearm the timer.

1: (Periodic): when the timer reaches zero, the Current Count Register is automatically reloaded with the value in the Initial Count Register, and the timer counts down again.

Timer Base: [Bits 19,18]

This field selects the time base input to be used by timer.

00: (Base 0) uses "CLKIN" as input;

01: (Base 1) uses "TMBASE";

10: (Base 2) uses the output of the divider (Base 2).

Bits [31:20] Bits [31:20] are Reserved. Should be written 0.

7.0 82489DX I/O UNIT REGISTERS

REGISTERS ADDRESSING SCHEME

The I/O Unit indirect addressing scheme uses two registers directly mapped into the processor's address space: the I/O Register Select register and the I/O Window register. The I/O register select register selects which I/O unit Register appears in the I/O Window register where it can be manipulated by software.

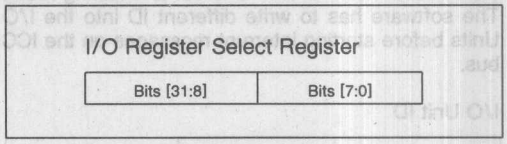


Figure 17. I/O Register Select Register

Bits [31:8]: Reserved. Should be written 0.

Bits [7:0]: I/O REGISTER SELECT: This register selects an 82489DX I/O unit register. The contents of the selected 32-bit register can be manipulated via the I/O Window Register. The I/O Register Select register is read-write by software.

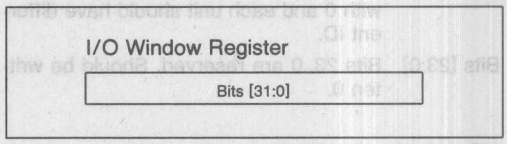


Figure 18. I/O Window Register

Bits [31:0] I/O WINDOW REGISTER: This register is mapped onto the I/O Unit's register selected by the I/O Register Select register. Readability/writability by software is determined by the I/O unit register that is currently selected.

The addresses (offsets to a platform-defined base address) of all registers are listed in the register summary section. Note that register offsets are aligned on 128-bit boundaries; in other words, registers are located only at every fourth 32-bit address. This eliminates the need for lane-steering glue logic when connecting the 82489DX's 32-bit data bus to a wider (64-bit and 128-bit) bus.

4

82489DX I/O UNIT CONFIGURATION

I/O Unit ID Register

Each 82489DX I/O Unit has a register that contains the I/O Unit's 8-bit ID. The I/O unit ID serves as a physical name of the 82489DX I/O Unit. It is used in arbitrating for ICC bus ownership when the I/O unit wants to access the ICC bus for sending any interrupt message. Unlike the local unit ID, the I/O unit ID is not latched-in from the address bus during hardware reset. The I/O unit ID is set to 0 during reset. The software has to write different ID into the I/O Units before starting interrupt messages on the ICC bus.

I/O Unit ID

Bits [31:24]	Bits [23:0]

Bits [31:24] I/O Unit ID:

The I/O unit ID serves as the physical "name" of the 82489DX unit used for arbitration purposes for the ICC bus usage. In a system with, say, four 82489DX, there are 4 Local Units and 4 I/O Units. All the 8 units should be assigned different ID. The IDs should start with 0 and each unit should have different ID.

Bits [23:0] Bits 23..0 are reserved. Should be written 0.

I/O Unit Version Register

Each 82489DX I/O Unit contains a hardware Version Register that identifies this 82489DX I/O unit version. This register is read only.

I/O Unit Version Register

Bits [31:24]	Bits [23:16]	Bits [15:8]	Bits [7:0]

Version: [Bits 7–0]

This is a version number that identifies this version. This field is hardwired and is read-only. Will be read as "1" for 82489DX.

Bits [15:8] Bits [15:8] are reserved.

Max Redir Entry: [Bits 23–16]

This is the entry number (0 being the lowest entry) of the highest entry in the Redirection Table. It is equal to the number of Interrupt Input Pins minus one of this I/O Unit. This field is hardwired and is read-only.

In the 82489DX I/O unit this is read as 15.

Bits [31:24] Bits [31:24] are reserved.

I/O UNIT INTERRUPT SOURCE REGISTERS

Redirection Tables

The Redirection Table has a dedicated entry for each interrupt input pin. Unlike IRQ pins of the 8259A, the notion of interrupt priority is completely unrelated to the position of the physical interrupt input pin on the 82489DX. Instead, software can decide for each pin individually what it wants the vector (and therefore the priority) of the corresponding interrupt to be. For each individual pin, the operating system can also specify whether the interrupt is signaled as edges or levels, as well as the destination and delivery mode of the interrupt. The information in the Redirection Table is used to translate the interrupt manifestation on the corresponding interrupt pin into an inter-82489DX message.

In order for a signal on an edge-sensitive Interrupt Input pin to be recognized as a valid edge (and not a glitch) the input level on the pin must remain asserted until the time 82489DX I/O Unit sends the corresponding message over the ICC bus. Only then will the source 82489DX be able to recognize a new edge on that Interrupt Input pin. That new edge will only result in a new invocation of the handler if its acceptance by the destination 82489DX causes the Interrupt Request Register bit to go from 0 to 1. (In other words, if the interrupt wasn't already pending at the destination.)

82489DX I/O unit has 16 Redirection Table entries. The layout of an entry in the Redirection Table is as follows:

Redirection Table Entry

Bits [31:17]	Bit 16	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bits [10:8]	Bits [7:0]

Vector (Bits [7:0]
Interrupt vector for this interrupt

Delivery Mode (Bits [10:8])
000: Fixed
001: Lowest Priority
010: <reserved>
011: <reserved>
100: NMI
101: Reset
110: <reserved>
111: ExtINT

Destination Mode (Bit 11)
0: Physical
1: Logical

Delivery Status (Bit 12)
0: Idle
1: Send Pending

Bit 13 Bit 13: Reserved. Should be written 0.

Remote IRR (Bit 14)
Reflects the Remote IRR bit
0: Remote IRR bit is clear.
1: Remote IRR bit is set.

Trigger Mode (Bit 15)
0: Edge
1: Level

Mask (Bit 16)
0: Not Masked
1: Masked

Bits [31:17] Reserved. Should be written 0.

DESCRIPTIONS

Vector: [Bits 7–0]
The vector field is an 8-bit field containing the interrupt for this interrupt.

Delivery Mode: [Bits 10–8]
The Delivery Mode is a 3-bit field that specifies how the 82489DXs listed in the destination field should act upon reception of this signal. Note that remote read is not supported for I/O device interrupts. Note that certain Delivery Modes will only operate as intended when used in conjuction with a specific Trigger Mode. These restrictions are indicated for each Delivery Mode.

000: (Fixed) means deliver the signal on the INT pin of all processors listed in the destination. Trigger Mode for "fixed" Delivery Mode can be edge or level.

001: (Lowest Priority) means deliver the signal on the INT pin of the processor that is executing at the lower priority among all the processors listed in the specified destination; Trigger Mode for "lowest priority" Delivery Mode can be edge or level.

100: (NMI) means deliver the signal on the NMI pin of all processors listed in the destination, vector information is ignored. A Delivery Mode equal to "NMI" requires a "level" Trigger Mode.

101: (Reset) means deliver the signal to all processors listed in the destination by asserting/deasserting the 82489DX's Reset output pin. All addressed 82489DXs' Local Units will assume their reset state but preserve their unit ID. One side effect of a unit message with Delivery Mode equal to "Reset" that results in a deassert of reset is that all 82489DXs' Local Units (whether listed in the destination or not) will reset their lowest-priority tie breaker arbitration ID to their unit ID (see the section on the ICC bus for details). A Delivery Mode of "Reset" requires a "level" Trigger Mode.

111: (ExtINT) means deliver the signal to the INT pin of all processors listed in the destination as an interrupt that originated in an externally connected (8259A-compatible) interrupt controller. The Local Unit receiving this interrupt will activate ExtINTA in response to this interrupt message. A Delivery Mode of "ExtINT" requires an "edge" Trigger Mode. (See the section on Compatibility for details.)

4

Destination Mode [Bit 11]

This field determines the interpretation of the Destination field.

0: (Physical Mode): in Physical Mode, a destination 82489DX Local Unit is identified by its unit ID. Bits 56 through 63 (8 MSB of the destination field) specify the 8-bit unit ID.

1: (Logical Mode): in Logical Mode, destinations are identified by matching on Logical Destination under the control of the Destination Format Register in each 82489DX Local Unit. The 32-bit Destination field is the logical destination.

Delivery Status: [Bit 12]

Delivery Status is a 1-bit field that contains the current status of the delivery of this interrupt. Two states are defined:

0: (Idle) means that there is currently no activity for this interrupt;

1: (Send Pending) indicates that the interrupt has been injected, but its delivery is temporarily held up by other recently injected interrupts that are in the process of being delivered; Delivery Status is software read-only; software writes to this field (as part of a 32-bit word) do not affect this bit.

Bit 13: Bit 13 is Reserved. Should be written 0.

Remote IRR: [Bit 14]

This bit is used for level triggered interrupts; its meaning is undefined for edge-triggered interrupts. Remote IRR mirrors the interrupt's IRR bit of the destination 82489DX Local Unit. When the value of the bit disagrees with the state of the Interrupt Input line, a unit message is automatically sent to make the destination's IRR both reflect the new state of the Interrupt Input line, and then the Remote IRR bit is updated to track its associated IRR bit. Remote IRR is software read-only; software writes to this bit do not affect it.

Trigger Mode: [Bit 15]

The Trigger Mode field indicates the type of signal on the interrupt pin that triggers an interrupt.

0: indicates edge sensitive,

1: indicates level sensitive.

Mask: [Bit 16]

Use this bit to mask injection of this interrupt.

0: indicates that injection of this interrupt is not masked. An edge or level on an interrupt pin that is not masked results in the delivery of the interrupt to the destination.

1: indicates that injection of this interrupt is masked. Edge-sensitive interrupts signaled on a masked interrupt Input pin are simply ignored (i.e., it is not delivered and is not held pending). Level-asserts or deasserts occurring on a masked level-sensitive pin are also ignored and have no side effects. As expected, changing the mask bit from unmasked to masked while the level remains asserted has the side effect of deasserting the level. It is software's responsibility to deal with the case where the Mask bit is set after the interrupt message has been sent but before the interrupt is dispensed to the processor.

Bits [31:17] Bits [31:17] are reserved. Should be written 0.

Destination

Bits [63:32]

Destination: If the Destination Mode of this entry is "Physical Mode", then the 8 MSB [bits 56 through 63] contain an 82489DX Local Unit ID. If Logical Mode, then the Destination field potentially defines a set of processors. The interpretation of the 32-bit destination field is further enabled by the Destination Format Register in the 82489DX Local Units.

8.0 ICC BUS DEFINITION

Physical Characteristics

The ICC bus is a 5-wire synchronous bus connecting all 82489DXs (all I/O Units and all Local Units). Four of these five wires are used for data transmissions and arbitration, and one wire is a clock. The description refers to the logical state of the ICC bus. Electrical levels are just inverse of the logical state described. For example, the section describes that the ICC bus is 0000 when not transmitting any message. This refers to logical state. Electrically, the ICC bus is 1111 when not transmitting any message.

The bus is electrically an open-drain connection providing for both bus use arbitration and arbitration for lowest priority. Being open-drain, the bus is run at a "comfortable" speed such that design-specific termination tuning is not required. Furthermore, each 82489DX receiving a message or participating in an arbitration must be given enough time in a single bus cycle to latch the bus and perform some simple logic operations on the latched information in order to determine whether the next drive cycle must be inhibited.

Note that it is likely in MP systems that additional processors be located on plug-in boards. Since the ICC bus would be part of the connector, the 82489DX to ICC bus connection is defined so that it can be electrically isolated using external drivers. The 82489DX has separate ICC bus input and output pins that can be connected externally to the 82489DX to either provide or not provide isolation.

The isolation can also be used to provide a hierarchical connection of ICC buses electrically supporting large numbers of processors. The number of 82489DXs supported using the hierarchical connection is limited only by ICC bus bandwidth. It should be noted that ICC bus output low current is just 4 mA.

Bus Arbitration

Arbitration (both for use of the bus and for determining the lowest priority 82489DX) depends on all 82489DX message units operating synchronously. To deal with the event where multiple agents start transmitting simultaneously, a distributed arbitration approach is used. Bus arbitration uses a small number of arbitration cycles in the ICC bus. During

4

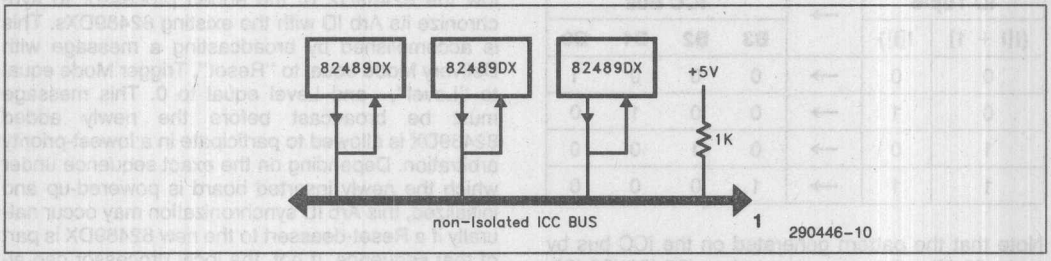


Figure 20. ICC Bus: Simple Direct Connection

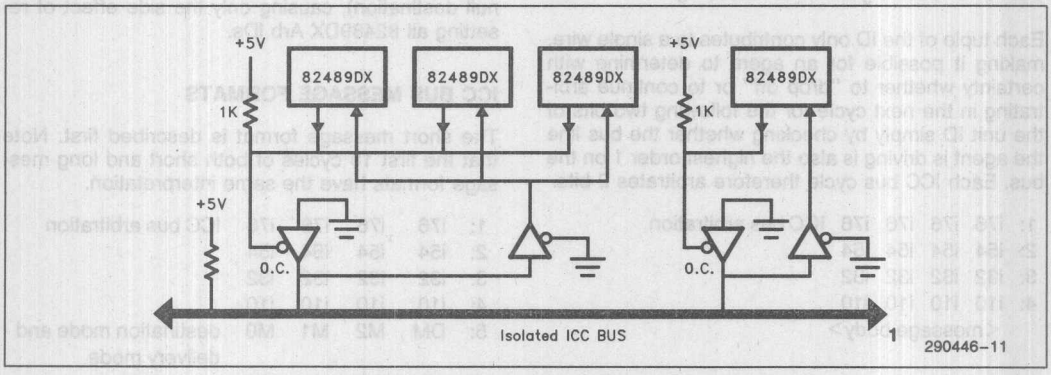


Figure 21. ICC Bits: Hierachical Connection

these cycles, arbitration losers progressively drop off the bus until only the winner remains transmitting. The winner then transmits its actual inter-unit message. Once the sending of a message (including bus arbitration) has started, any possible contender must suppress transmission until enough cycles have elapsed for the message to be fully sent. The number of message cycles depends on the type of message being sent.

A bus arbitration cycle starts by the agent driving its unit ID on the ICC bus. High-order ID bits are driven first, successive cycles proceeding to the low bits of the ID. All losers in a given cycle drop off the bus, using every subsequent cycle as a tie breaker for the previous cycle. By the time all arbitration cycles are completed, there will be only a single agent left driving the bus.

The 8-bit unit ID (I7 I6 I5 I4 I3 I2 I1 I0) is chopped up in successive groups of 2 bits (I7 I6)(I5 I4)(I3 I2)(I1 I0). Each of these tuples is first decoded before driving them on the bus. The 0s and 1 indicate logical levels and not signal levels. The ICC bus is 0000 when not transmitting any message. The decoding used is:

ID Tuple		→	ICC Bus			
(I[i + 1]	I[i])		B3	B2	B1	B0
0	0	→	0	0	0	1
0	1	→	0	0	1	0
1	0	→	0	1	0	0
1	1	→	1	0	0	0

Note that the pattern generated on the ICC bus by tuple (I3 I2) will be represented as i32 i32 i32 i32. The lower case signifies this encoding.

Each tuple of the ID only contributes to a single wire, making it possible for an agent to determine with certainty whether to "drop off" or to continue arbitrating in the next cycle for the following two bits of the unit ID simply by checking whether the bus line the agent is driving is also the highest order 1 on the bus. Each ICC bus cycle therefore arbitrates 2 bits.

1:	i76	i76	i76	i76	ICC bus arbitration
2:	i54	i54	i54	i54	
3:	i32	i32	i32	i32	
4:	i10	i10	i10	i10	
	<message body>				

Lowest-Priority Arbitration

Arbitration is also used to find the 82489DX Local Unit with the lowest processor priority. Lowest-priority arbitration uses the value of the 82489DX's Processor Priority value appended with an 8-bit Arbitration ID (Arb ID) to break ties in case there are multiple units executing at the lowest priority.

Using the constant 8-bit unit ID as the Arb ID has a tendency to skew symmetry since it would favor 82489DXs with low ID values. An 82489DX Local Unit's Arb ID is therefore not the unit ID itself but is derived from it. At reset, an 82489DX Local Unit's Arb ID is equal to its unit ID. Each time a message is broadcast over the ICC bus in lowest priority mode, all 82489DX Local Units increment their Arb ID by one, which gives them a different Arb ID value for the next arbitration. The Arb ID is then endian-reversed (LSB becomes MSB, etc.) to ensure better rotation of which 82489DX gets to have the lowest Arb ID next time around. The reversed Arb ID is then decoded to generate arbitration signals on the ICC bus as described above.

To support hot insertion of processor boards in a running MP system, a mechanism is provided to allow the 82489DX of the added processor to synchronize its Arb ID with the existing 82489DXs. This is accomplished by broadcasting a message with Delivery Mode equal to "Reset", Trigger Mode equal to "Level", and Level equal to 0. This message must be broadcast before the newly added 82489DX is allowed to participate in a lowest-priority arbitration. Depending on the exact sequence under which the newly inserted board is powered-up and initialized, this Arb ID synchronization may occur naturally if a Reset-deassert to the new 82489DX is part of that sequence. If not, the local processor can always send this as an inter processor interrupt (with a null destination), causing only the side effect of resetting all 82489DX Arb IDs.

ICC BUS MESSAGE FORMATS

The short message format is described first. Note that the first 19 cycles of both short and long message formats have the same interpretation.

1:	i76	i76	i76	i76	ICC bus arbitration
2:	i54	i54	i54	i54	
3:	i32	i32	i32	i32	
4:	i10	i10	i10	i10	
5:	DM	M2	M1	M0	destination mode and delivery mode
6:	"0"	"0"	L	TM	control bits
7:	V7	V6	V5	V4	vector
8:	V3	V2	V1	V0	

9:	D31	D30	D29	D28	destination
10:	D27	D26	D25	D24	
11:	D23	D22	D21	D20	
12:	D19	D18	D17	D16	
13:	D15	D14	D13	D12	
14:	D11	D10	D09	D08	
15:	D07	D06	D05	D04	
16:	D03	D02	D01	D00	
17:	C	C	C	C	checksum for cycle 5 through 16
18:	"1"	"1"	"1"	"1"	post amble
19:	A	A	A	A	accept (1000 if OK, 1110 if preempt, else error)
20:	"0"	"0"	"0"	"0"	idle 1
21:	"0"	"0"	"0"	"0"	idle 2

Cycles 1 through 4 are bus arbitration as described earlier. Cycle 5 (DM M2 M1 M0) is the Destination Mode which is 0 for Physical mode and 1 for Logical Mode, and the Delivery Mode of the message. The encoding used for the Delivery Mode in the message is identical to the encoding used for the Delivery Mode in the Redirection Table, Local Vector Table, and Interrupt Command Register.

M2	M1	M0	Delivery Mode
0	0	0	Fixed
0	0	1	Lowest Priority
0	1	0	<reserved>
0	1	1	Remote Read
1	0	0	NMI
1	0	1	Reset
1	1	1	ExtINT

Cycle 6 contains the Control Bits of the message. The control bits are:

- TM (Trigger Mode): indicates whether this message corresponds to an edge or level;
- L (Level): indicates whether this is an Assert or a Deassert of a "level" signal. L is undefined when TM is edge.

6: "0" "0" L TM Control Bits
TM = Trigger Mode (0 = edge, 1 = level)
L = Level (0 = deassert, 1 = assert)

The length of the message is derived from the Delivery Mode, the Control Bits, and the Accept cycle of the message.

TM/L (AAAA)

	Edge	Level = Assert	Level = Deassert
Fixed	Short	Short	Short
Lowest Priority	Short (1110)	Short (1110)	Short (1110)
	Long (1000)	Long (1000)	Short
Remote Read	Long	Long	Short
NMI	Short	Short	Short
Reset	Short	Short	Short
ExtINTA	Short	Short	Short

Cycles 7 and 8 are the 8-bit interrupt vector. The vector is only defined for Delivery Modes Fixed, and Lowest-priority. For Delivery Mode of "Remote Read", the vector field contains the address of the register to be read remotely.

If DM is 0 (physical mode), then cycles 9 and 10 are the unit ID and cycles 11 through 16 are zero. If DM is 1 (logical mode), then cycles 9 through 16 are the 32-bit Destination field. The interpretation of the logical mode 32-bit Destination field is performed by the Local Units using the Destination Format Register. The sending 82489DX knows whether it should (incl) or should not (excl) respond to its own message.

Cycle 17 is a checksum over the data in cycles 5 through 16. The checksum is computed by adding all 4-bit quantities of cycles 5 through 16, feeding carry out of the MSB back into the LSB. This protects the data in these cycles against transmission errors. The (single) 82489DX driving the message provides this checksum in cycle 17.

Cycle 18 is a post amble cycle driven as 1111 by the sending 82489DX allowing all 82489DXs to perform various internal computations based on the information contained in the received message. One of the computations takes the computed checksum of the data received in cycles 5 through 16 and compares it against the value in cycle 17. If any 82489DX computes different checksum than the one passed in cycle 17, then that 82489DX will signal an error on the ICC bus in cycle 19 by driving it as 1111. If this happens, all 82489DXs will assume the message was never sent and the sender must try sending the message again, which includes re-arbitrating for the ICC bus. In lowest priority delivery when the interrupt has a focus processor, the focus 82489DX will signal this by driving 1110 during cycle 19. This tells all the other 82489DXs that the interrupt has been accepted, the 82489DXs is preempted, and short message format is used. All (non-focus) 82489DXs will drive 1000 in cycle 19. Under lowest priority mode, 1000 implies that the interrupt currently has no focus processor and that priority arbitration is required to complete the delivery. In that case, long message format is used. If cycle 19 is 1000 for non Lowest Priority mode, then the message has been accepted and is considered sent.

	19:EEEE
1000	OK
1110	preempt
<others>	error (drive error as 1111)

When an 82489DX detects and reports an error during the error cycle, that 82489DX will simply listen to the bus until it encounters two consecutive idle (0000) cycles. These two idle cycles indicate that the message has passed and a new message may be started by anyone. This allows an 82489DX that got itself out of cycle on the ICC bus to get back in sync with the other 82489DXs.

Long Message Format

Cycles 1 through 19 of the long message format are identical to cycles 1 through 19 of the short message format. As mentioned, long message format is used in two cases:

(1) Lowest Priority delivery when the interrupt does not have a focus. Cycles 20 through 27 are eight arbitration cycles where the destination 82489DXs determine the one 82489DX with lowest processor priority/ARB ID value.

(2) Remote Read messages. Cycles 20 through 27 are the 32-bit content of the remotely read register. This information is driven on the bus by the remote 82489DX.

Cycle 28 is an Accept cycle. In lowest priority delivery, all 82489DXs that did not win the arbitration (including those that did not participate in the arbitration) drive cycle 28 with 1000 (co accept), while the winner 82489DX drives 1111. If cycle 28 reads 1111, then all 82489DXs know that the interrupt has been accepted and the message is considered delivered. If cycle 28 reads 1100 (or anything but 1111 for that matter), then all 82489DXs assume the message was unaccepted or an error occurred during arbitration. The message is considered undelivered, and the sending 82489DX will try delivering the message again.

For Remote Read messages, cycle 28 is driven as 1100 by all 8 2489DXs except the responding remote 82489DX, who drives the bus as 1111 in case it was able to successfully supply the requested data in cycles 20 through 27. If cycle 28 reads 1111 the data in cycles 20 through 27 is considered valid; otherwise, the data is considered invalid. The source 82489DX that issued the Remote Read uses cycle 28 to determine the state of the Remote Read Status field in the Interrupt Command Register (valid or invalid). In any case, a Remote Read request is always successful (although the data may be valid or invalid) in that a Remote Read is never retried. The reason for this is that Remote Read is a debug feature, and a "hung" remote 82489DX that is unable to respond should not cause the debugger to hang.

Cycles 29 and 30 are two idle cycles. The ICC bus is available for sending the next message at cycle 31. The two idle cycles at the end of both short and long messages, together with non zero (i.e., non idle) encoding for certain other bus cycles allow an ICC bus agent that happens to be out of phase by one cycle to sync back up in one message simply by waiting for two consecutive idle cycles after reporting its checksum error. This makes use of the fact that valid arbitration cycles are never 0000.

1:	i76	i76	i76	i76	ICC bus arbitration
2:	i54	i54	i54	i54	
3:	i32	i32	i32	i32	
4:	i10	i10	i10	i10	
5:	DM	M2	M1	M0	delivery mode
6:	"0"	"0"	L	TM	control bits
7:	V7	V6	V5	V4	vector
8:	V3	V2	V1	V0	
9:	D31	D30	D29	D28	destination
10:	D27	D26	D25	D24	
11:	D23	D22	D21	D20	
12:	D19	D18	D17	D16	
13:	D15	D14	D13	D12	
14:	D11	D10	D09	D08	
15:	D07	D06	D05	D04	
16:	D03	D02	D01	D00	
17:	C	C	C	C	checksum for cycles 5 through 16
18:	"1"	"1"	"1"	"1"	postamble
19:	A	A	A	A	accept (1000 if OK, 1110 if preempt, else error)
20:	p76	p76	p76	p76	lowest priority arbitration or 32 bits of remote register
21:	p54	p54	p54	p54	processor priority
22:	p32	p32	p32	p32	
23:	p10	p10	p10	p10	
24:	a76	a76	a76	a76	
25:	a54	a54	a54	a54	arbitration ID
26:	a32	a32	a32	a32	
27:	a10	a10	a10	a10	
28:	A	A	A	A	accept
29:	"00"	"0"	"0"	"0"	idle1
30:	"0"	"0"	"0"	"0"	idle2

9.0 HARDWARE TIMINGS

This section covers the following:

— Timing Diagram Notation

— 82489DX Register Access Timing Diagrams with Descriptions

A block diagram of the configuration of the CPU module of a MP system is shown. This in no way is intended to be a complete representation of 486/Intel Cache/Intel Cache Controller connections. It is intended to show all the 82489DX connections, and how they connect to other components on and off the module. This module has arbitrarily been drawn with a 64-bit data bus to show how the expanded address space architecture fits. The unit can be similarly attached to either a 32-bit or 128-bit data bus, with total transparency to shrink-wrap software.

In this configuration, the 82489DX uses the same clock source as the processor and cache. However, it is quite possible to consider 82489DX as a memory bus device and hence supply 82489DX with the memory bus clock, which can be slower than the CPU module clock frequency.

In the configuration shown, the processor's INT and NMI pins could be supplied by other source to allow for the possibility that 82489DX can be totally bypassed if desired, by allowing those signals to be driven from off the module while the 82489DX is disabled. The reset signal generated by the 82489DX goes to the MBC (memory bus controller) which is required to drive configuration lines at reset time. This would probably be configured as a "warm" reset by the MBC.

A future version of cache controller may generate the chip select for 82489DX at a fixed memory location of hexFEE00000. By having the cache controller to provide the chip select signal, it would encourage a standard mapping for 82489DX address space. In some MBC designs, this signal should be connected to the MBC since 82489DX cycles limit bus pipelining by constraining how soon the next bus cycle can come. The 82489DX chip select can be generated by the MBC completely.

The address, data and most of the bus control signals share the respective bus with cache and cache controller. The block diagram shows attachment for only 6 address lines: A4–A9. A10 should be 0. This is all the 82489DX needs for operation, however, if the address lines are used to initialize 82489DX local ID at reset time, 8 address lines are required, A3–A10.

INTERFACING TO THE ICC BUS

The 82489DX has separate ICC bus input and output pins to facilitate using external drivers. The ICC bus input pins (MBI0-3) are TTL-level compatible CMOS inputs. The output pins (MBO0–3) are open-drain pins which required external pull-ups. The open-drain output buffers are small buffers with:

Sink current of < 4 mA. Special consideration must be exercised when driving large capacitive loads or long transmission lines. The pull-up resistor and the capacitive load constitute RC time constant that will affect the output transition times. This in turn will limit the operating frequency of the ICC bus.

When designing in the ICC bus, one needs to consider the loads that each 82489DX will be driving and whether external drivers should be used. In most situations, the ICC bus driven high (MBO pins pulled high by the external pull-up resistors) poses the most challenge. Simulating the target design on an electrical simulator (such as SPICE) will help greatly. as shown in the following examples.

First Order Buffer Models

Figure 21a and 21b are first order input buffer and output buffer models of the MBI and MBO pins. The open-drain of the MBO is modeled as a switch as the primary interest here is the MBO pins going high. These models can be used on SPICE simulations to obtain first order behaviors. The parameters for these models are as follows:

Cp (package capacitance) = 3 pF
Lp (package inductance) = 15 nH
Rb (bond wire resistance) = 0.08Ω
Ci (input buffer capacitance) = 3 pF
Co (output buffer capacitance) = 6 pF
Ro (output buffer impedance) = 30Ω–80Ω

MBO Pull-up Resistor

To minimize the RC time constant, one would like to use the smallest pull-up resistor value possible. The MBO pins has a worst case Iol-spec of 4 mA and V_{CC} = 4.75V. This translates to a minimum pull-up of about 1 KΩ. Where stronger drive is needed (smaller pull-up resistors), external drivers must be used.

Driving Lumped Capacitance

In systems where external drivers are not used, the MBI pins will be tied to the MBO pins. Figure 21d is a SPICE simulation of the MBO output with a 1 KΩ pull-up driving lumped capacitive loads from 10 pF to 150 pF.

At a load of 50 pF, it takes about 30 ns to charge up to 2V. At 100 pF, it takes an additional 25 ns. Figure 21d can be used to estimate the loading delay at different lumped capacitive loads.

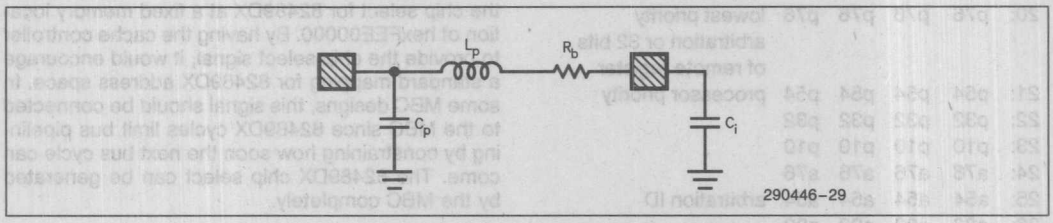


Figure 21a. First Order Input Buffer for MBI Pins

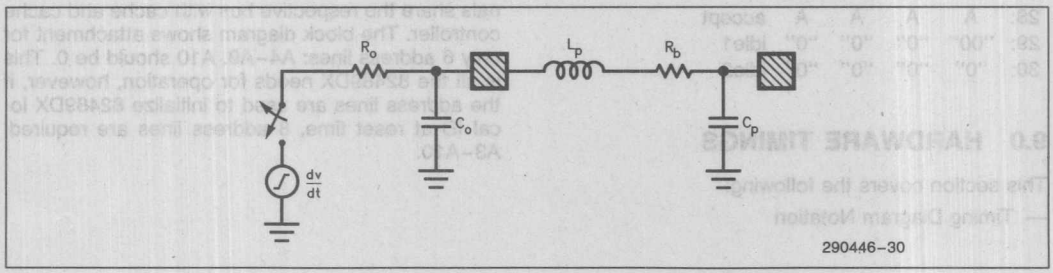


Figure 21b. First Order Open-Drain Output Buffer for MBO Pins

In real systems, the loads are made up of lumped capacitance and transmission lines. More accurate results can be obtained using transmission line models.

Driving Transmission Lines

Two device model

In this example the ICC bus is a signal line on an FR-4 printed circuit board. The line width is 6 mils. Line length of 12 inches and 18 inches are modeled. The FR-4 PC board has the following characteristics:

resistivity =	0.6 mΩ/sq. (0.1Ω/inch for 6 mil width)
inductance =	60 pH/sq. (10 nH/inch for 6 mil width)
capacitance =	0.55 nF/sq. in. (3.3 pF/inch for 6 mil width)

The ICC bus is shared by two 82489DXs, one at each end. The ICC bus is modeled as a transmission line. For the simulation, only one of the 82489DX is driving. A pull-up resistor of 2 KΩ is used at each end (1 KΩ equivalent value) as shown in Figure 21e. Figure 21f shows the signals at each end of the 12 inch transmission line. Trace 1 is the wave form at the driven end and trace 2 is the signal at the receiving end of the line. The 2 ns delay between the two signals is the propagation delay (or flight time) through the 12 inch transmission line. It takes about 35 ns for the voltage to charge up to 2V.

Figure 21g shows the received signal with different line length and with additional lumped capacitance. Trace 1 is for 12 inch only. Trace 2 is for 12 inch with additional 20 pF lumped capacitance to represent interconnect socket capacitance. Trace 3 is for 18 inch plus 20 pF. The presence of the 20 pF at each end of the 12-inch transmission line increases the delay time by 20 ns at 2V.

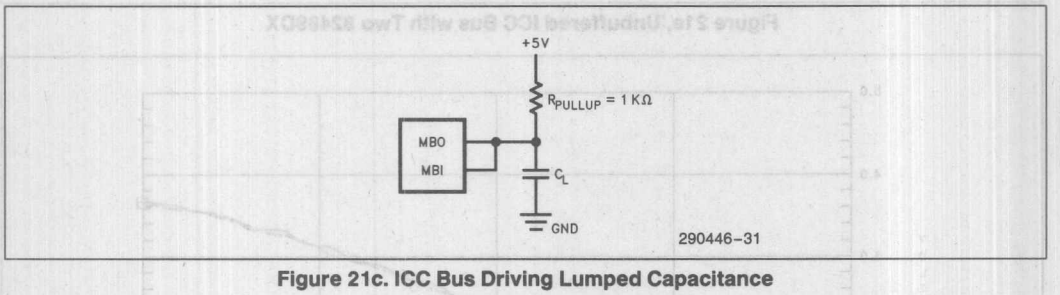


Figure 21c. ICC Bus Driving Lumped Capacitance

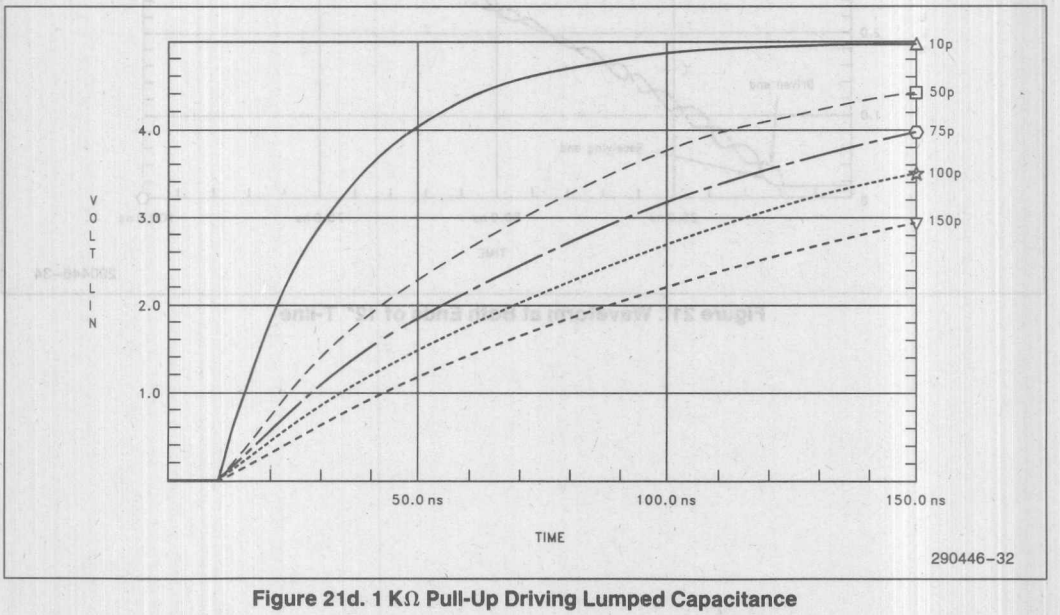


Figure 21d. 1 KΩ Pull-Up Driving Lumped Capacitance

Four Device Model

In this example (Figure 21h), the ICC bus is a 12-inch transmission line with four 82489DXs connected at 4 inch intervals. The loading at each junction consisted of the MBI and MBO buffers and a 20 pF lumped capacitance. 2 KΩ pull-ups are at each end of the transmission line.

As shown in Figure 21i, it takes more than 90 ns for the signal level at both ends to reach 2V.

One way to improve the low to high transition time is to use a stronger pull-up (smaller resistor value) which is possible using external line drivers with their larger current drive capabilities.

Figure 21j shows the difference in output when the model is used with 300Ω pull-ups at each end of the transmission line.

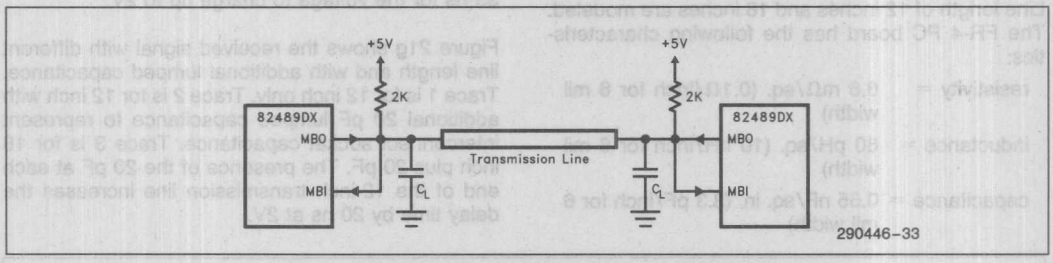


Figure 21e. Unbuffered ICC Bus with Two 82489DX

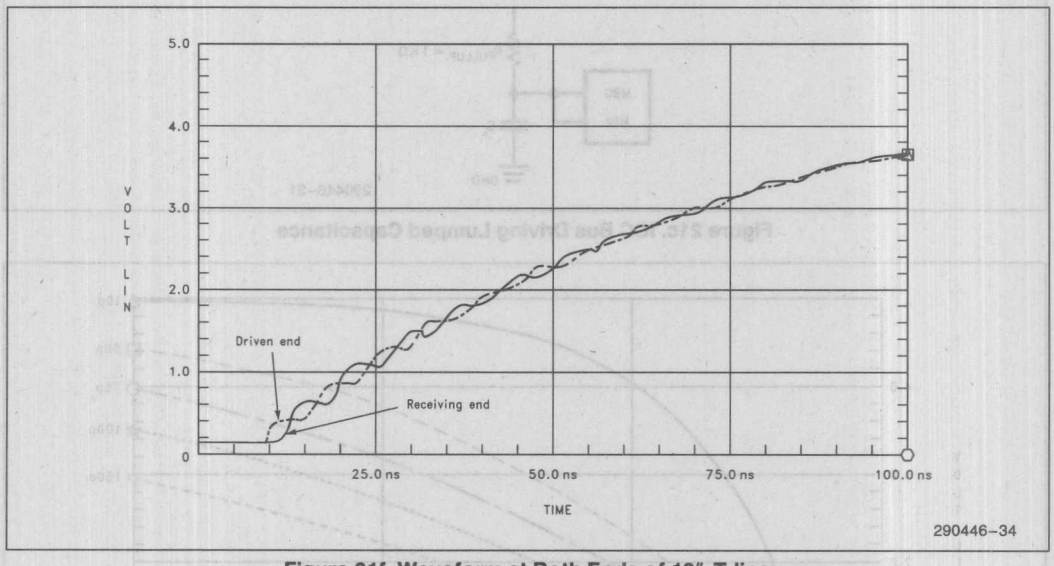


Figure 21f. Waveform at Both Ends of 12" T-line

External Drivers/Buffered ICC bus

The 82489DX has separate ICC Bus input (MBI) and output (MBO) pins that can be connected to external line drivers in systems that has appreciable loading on the ICC Bus or where modularity of the bus is needed.

Figure 21k is a typical implementation using external drivers with tri-state outputs. Drivers such as 74F125 or its equivalent can be used. The drivers should be placed as close to the MBO pins as possible. The input buffer on MBI is optional depending on the users ICC Bus scheme. The total delay through the drivers, buffers, transmission line, clock skews etc. must be calculated to ensure that all the ICC bus timing requirements are met.

A hierarchical bus connection can also be used in applications that cannot afford driver/buffer per unit and where bus loading are localized in cluster groups. Figure 21l shows such a connection where each cluster group is connected directly and drivers are used to connect to other clusters. Each cluster group is assumed to be close together physically with small loading on the local ICC bus.

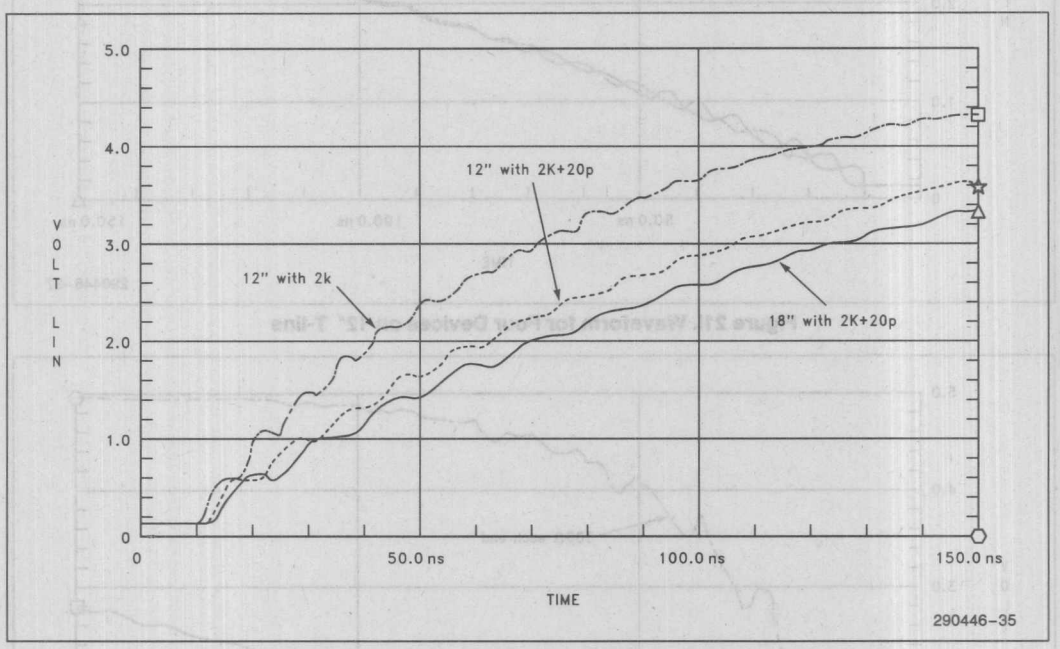


Figure 21g. Waveform at End of T-line with Load

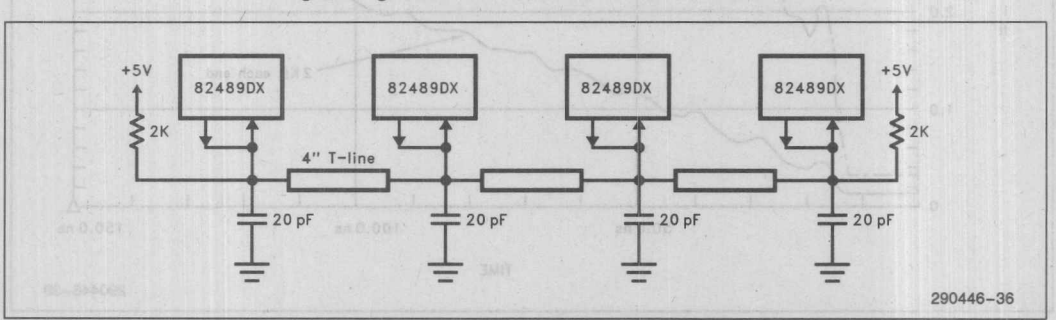


Figure 21h. Four 82489DX Configuration

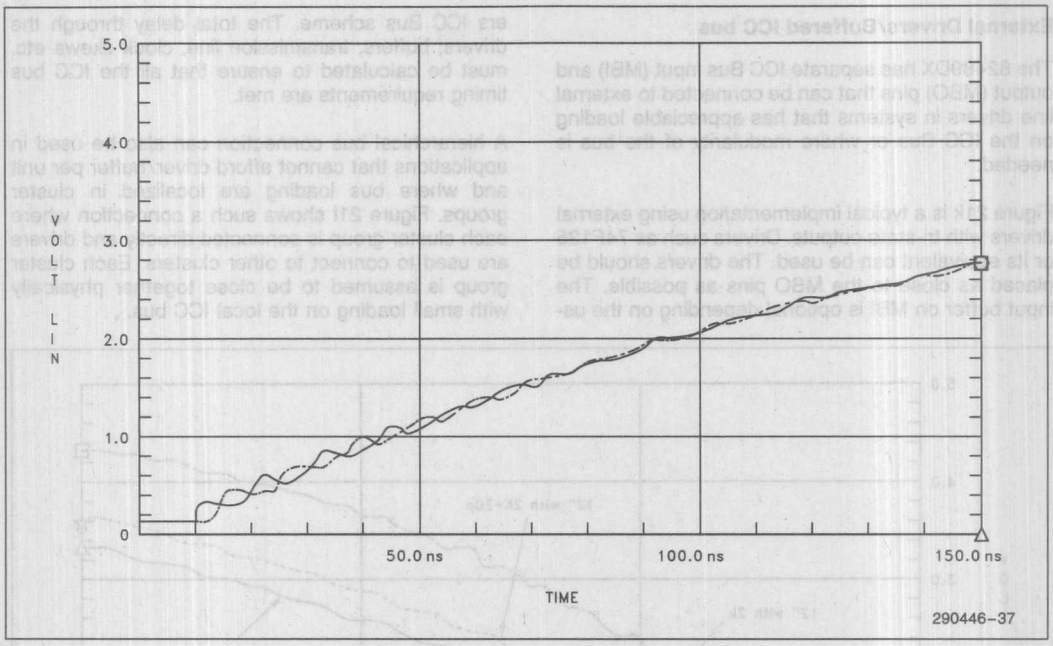


Figure 21i. Waveform for Four Devices on 12" T-line

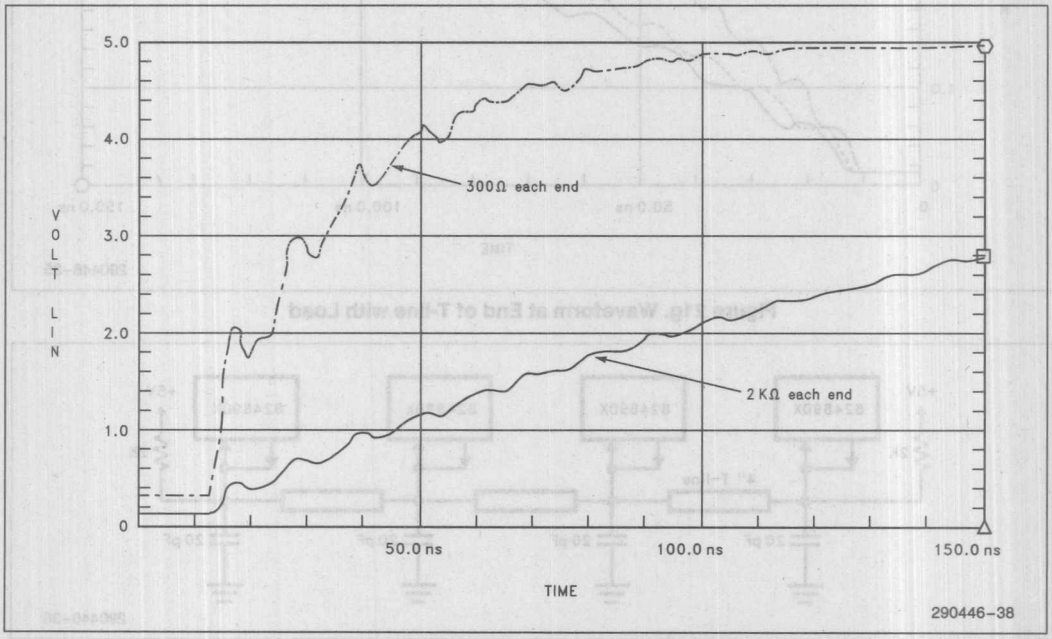


Figure 21j. Waveform with Different Pull-Ups

Transmission Line Termination

As with all high speed designs, one has to consider transmission line effects on signals, especially clock signals. Even though the ICC bus clock, ICLK is usually operated in the 10 MHz range, one has to consider proper transmission line termination also for short rise times. Figure 21m shows the ICLK wave form at the end of a 12 inch T-line when driven by a clock generator with and without series matched termination.

Series termination should not be used for the ICC bus data lines (MBO). The combination of the pull-up resistor and series resistor would degrade the output low voltage, Vol. For example, with a pull-up of 300Ω and a series termination of 50Ω at each end, the Vol voltage at the receiving end would be at 1.55V if the driving end is at 0.4V (see Figure 21n).

ICC BUS Operating Frequency

The 82489DX ICC BUS has a design target of operating up to 16 MHz (62 ns period). As shown in the examples above, the MBO low-to-high transition times are strongly dictated by the loads and the pull-ups used. This will in turn affect the maximum operating frequency of ICLK.

In general, the minimum period is the larger of 62 ns or MBO-to-MBI low data time or MBO-to-MBI high data time.

MBO-to-MBI low time =
(ICLK skew + MBO valid low delay + T-line prop.delay + ext. buffer delay + MBI setup time)

MBO-to-MBI high time =
(ICLK skew + MBO Hi-Z delay + pull-high time + T-line prop.delay + ext. buffer delay + MBI setup time)

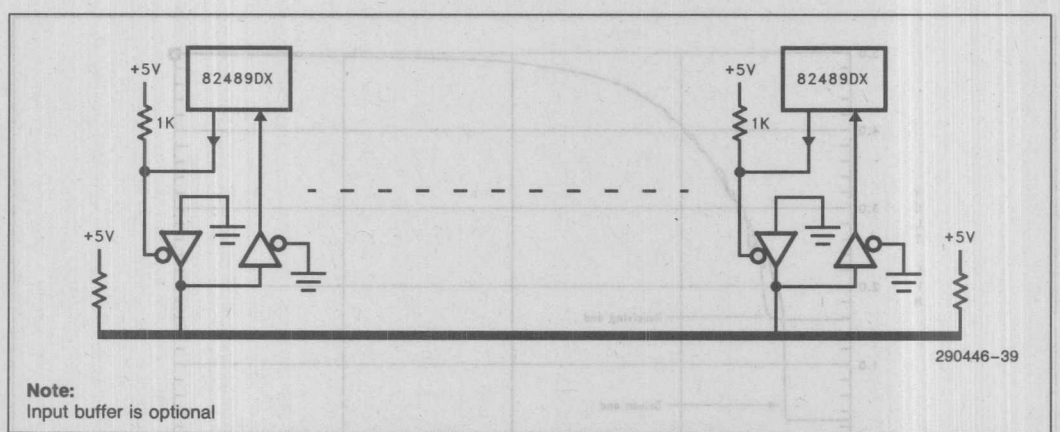
Maximum MBO valid low delay = 50 ns
Maximum MBO H-Z delay = 15 ns
MBI minimum setup time = 8 ns

In the example shown earlier where two 82489DXs are at each end of a 12-inch T-line with no other loads, the pull-high time to 2V is 35 ns (trace 1 in Figure 21g). If the ICLK skew is 2 ns, then this configuration can operate to 62 ns period or 16 MHz.

If the same configuration has additional 20 pF loads at each end, then the pull-high time is 55 ns (trace 2 in Figure 21g). The maximum frequency decreases to 12 MHz (82 ns period).

In the four device model discussed earlier, where the ICC Bus is unbuffered, the pull-high time is 90 ns (Figure 21j). The operating frequency will be less than 8 MHz (117 ns period). If external buffers are used (whereby allowing use of 300Ω pull-up) and assuming the external buffers have delays of 10 ns, the operating frequency is limited by the MBO-to-MBI low time of 72 ns or 14 MHz.

NOTE:
Each application is unique in its configuration and loading on the ICC Bus. The above examples highlighted some of the factors that need to be considered. It is important to do electrical simulation to ascertain if the propose implementation is viable before committing to the printed circuit board.



Note:
Input buffer is optional

Figure 21k. External Driver/Buffer Implementation

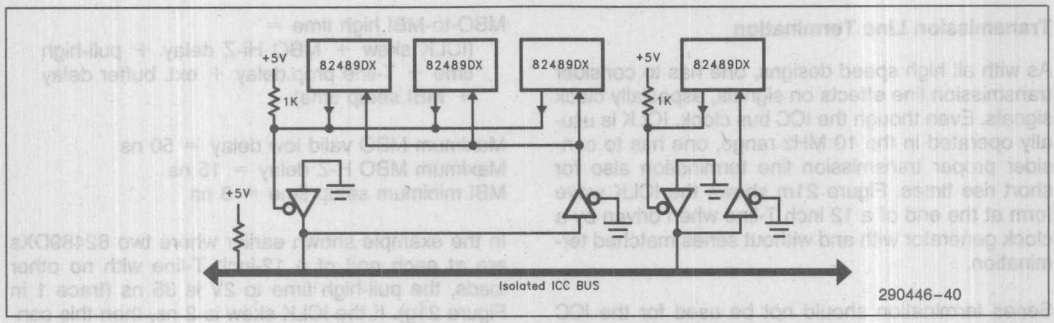


Figure 21l. ICC Bus: Hierarchical Implementation

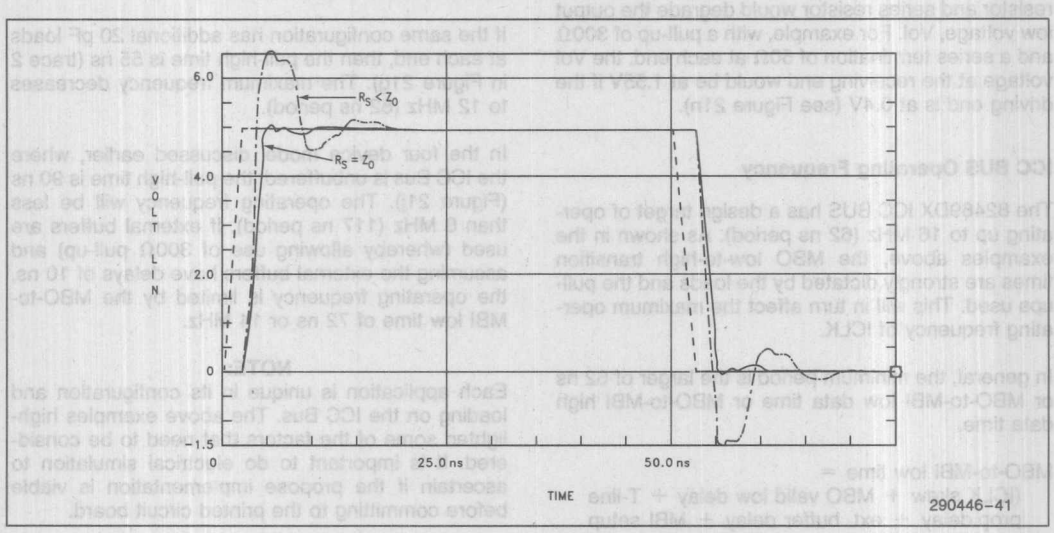


Figure 21m. ICLK Waveform on 12" T-line

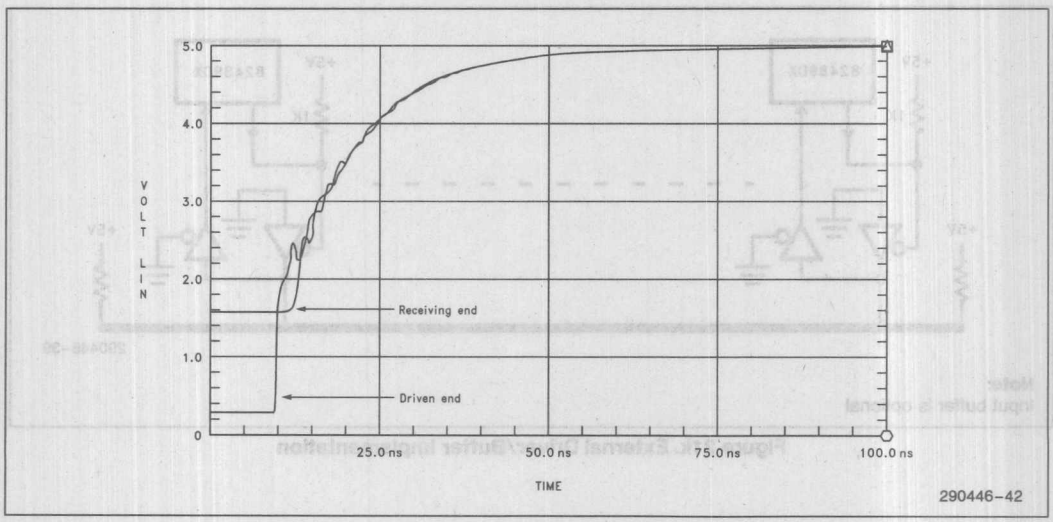


Figure 21n. Effect of Series Termination on MBO VOL

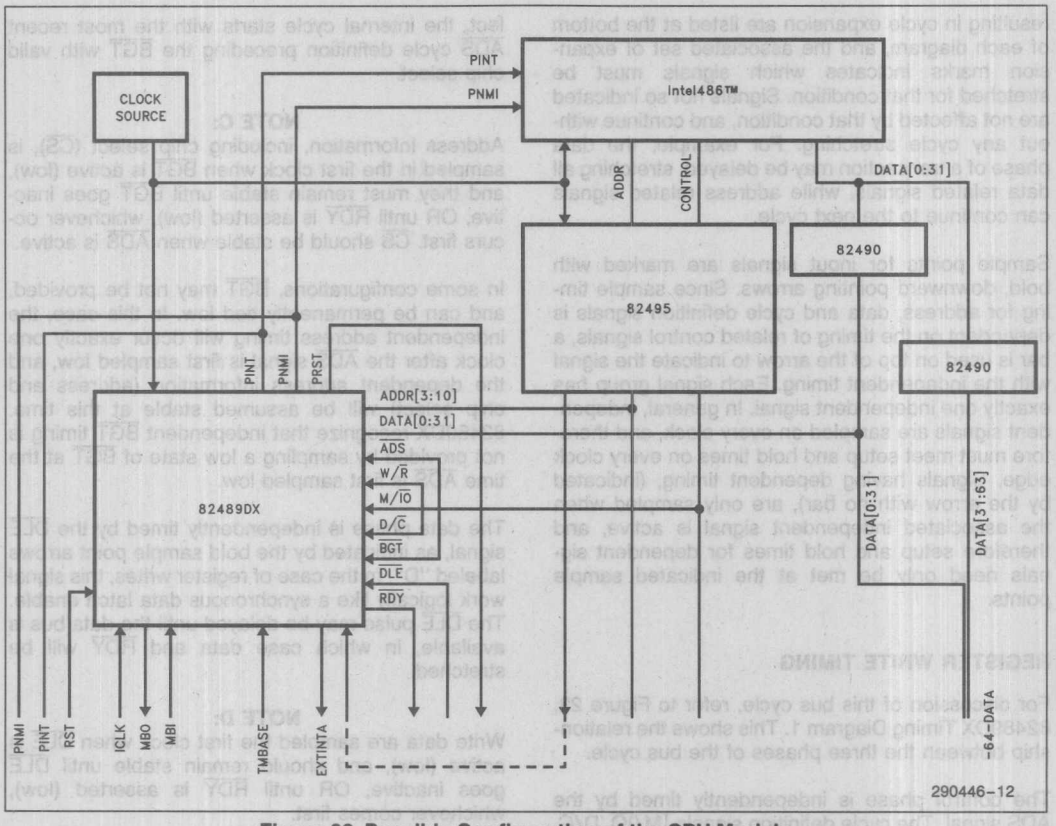


Figure 22. Possible Configuration of the CPU Module

9.1 82489DX Register Access Timing

This section provides descriptions of the four basic cycle timings for the 82489DX, which are:

— Register Write
— Register Read
— Interrupt Acknowledge
— Reset

In addition, Inter-Unit (nibble) bus timings are presented.

Register access occurs in three distinct phases:

1. Control Phase,
2. Address Phase and
3. Data Phase.

They always occur in this order, although in some cases address and data phases can occur in the same clock cycle, as will be seen in the diagrams.

NOTE:
As mentioned previously, the clock signal in all these timing diagrams is assumed to be the processor clock.

TIMING DIAGRAM NOTATION

The 82489DX bus (register access) interface is synchronous. Unless otherwise noted, setup and hold times for inputs, and delay times for outputs are measured with respect to a rising clock edge. Therefore the timing diagrams contain very few construction lines to identify timing parameters and ones that exist are explicitly discussed. High and low level for signals are obvious; dashed lines at the middle of the signal range indicate a tristate output buffer in high impedance mode; hashed or cross-hatched lines indicate a "don't care" state for inputs.

Cycle expansion marks—short curved lines breaking the waveform—appear throughout. These appear in sets or groups which are identified by construction arrows and/or vertical column alignment. Conditions

resulting in cycle expansion are listed at the bottom of each diagram, and the associated set of expansion marks indicates which signals must be stretched for that condition. Signals not so indicated are not affected by that condition, and continue without any cycle stretching. For example, the data phase of a transaction may be delayed, stretching all data related signals, while address related signals can continue to the next cycle.

Sample points for input signals are marked with bold, downward pointing arrows. Since sample timing for address, data and cycle definition signals is dependent on the timing of related control signals, a bar is used on top of the arrow to indicate the signal with the independent timing. Each signal group has exactly one independent signal. In general, independent signals are sampled on every clock, and therefore must meet setup and hold times on every clock edge. Signals having dependent timing, (indicated by the arrow with no bar), are only sampled when the associated independent signal is active, and therefore setup and hold times for dependent signals need only be met at the indicated sample points.

REGISTER WRITE TIMING

For discussion of this bus cycle, refer to Figure 23, 82489DX Timing Diagram 1. This shows the relationship between the three phases of the bus cycle.

The control phase is independently timed by the $\overline{ADS}$ signal. The cycle definition signals [M/$\overline{IO}$, D/$\overline{C}$, are dependently sampled with $\overline{ADS}$ as indicated by the bold sample point arrows labeled "C". The cycle definition signals will be sampled in the first clock when $\overline{ADS}$ is active (low). The control signal should remain stable until $\overline{ADS}$ goes inactive. For any valid 82489DX cycle, the memory bus controller should ensure that the $\overline{ADS}$ pulse for a subsequent bus cycle is NOT presented until after the 82489DX asserts its $\overline{RDY}$ pin (low) as shown.

The address phase is independently timed by the ($\overline{BGT}$) signal, as indicated by the bold sample point arrows labeled "A". This signal is actually used an address latch enable, however, its name is intended to imply that in most cases it can be directly driven by the Intel cache controller signal of the same name. The $\overline{BGT}$ pulse may be delayed until the address bus is available, in which case all address and data phase signals will be stretched. Note that $\overline{DLE}$ must not occur before $\overline{BGT}$. 82489DX does not start the internal cycle until $\overline{BGT}$ is recognized with the appropriate chip select signal, If multiple $\overline{ADS}$ has been issued without $\overline{BGT}$ and a valid chip select, the internal cycle starts with the most recent $\overline{ADS}$ cycle definition preceding the $\overline{BGT}$ with valid chip select.

NOTE C:
Address information, including chip select ($\overline{CS}$), is sampled in the first clock when $\overline{BGT}$ is active (low), and they must remain stable until $\overline{BGT}$ goes inactive, OR until $\overline{RDY}$ is asserted (low), whichever occurs first. $\overline{CS}$ should be stable when $\overline{ADS}$ is active.

In some configurations, $\overline{BGT}$ may not be provided, and can be permanently tied low. In this case, the independent address timing will occur exactly one clock after the $\overline{ADS}$ signal is first sampled low, and the dependent address information (address and chip select) will be assumed stable at this time. 82489DX recognize that independent $\overline{BGT}$ timing is not provided by sampling a low state of $\overline{BGT}$ at the time $\overline{ADS}$ is first sampled low.

The data phase is independently timed by the $\overline{DLE}$ signal, as indicated by the bold sample point arrows labeled "D". In the case of register writes, this signal work logically like a synchronous data latch enable. The $\overline{DLE}$ pulse may be delayed until the data bus is available, in which case data and $\overline{RDY}$ will be stretched.

NOTE D:
Write data are sampled the first clock when $\overline{DLE}$ is active (low), and should remain stable until $\overline{DLE}$ goes inactive, OR until $\overline{RDY}$ is asserted (low), whichever comes first.

In some configurations, $\overline{DLE}$ may not be provided, and can be permanently tied low. In this case, the independent data timing will occur exactly one clock after the $\overline{ADS}$ signal is first sampled low, OR on the same clock as $\overline{BGT}$ is first sampled low, whichever occurs later. The data bus will be assumed stable at this time. 82489DX recognize that independent $\overline{DLE}$ timing is not provided by sampling a low state of $\overline{DLE}$ at the time $\overline{ADS}$ is first sampled low.

Cycle completion is signaled by the $\overline{RDY}$ signal. Its relative positioning on any of the timing diagrams does NOT imply the number of clock cycles required for an access. $\overline{RDY}$ is delayed as needed in order for the 82489DX to complete the cycle. It is then asserted (low) for one clock cycle and then deasserted. Again, the next $\overline{ADS}$ cannot start until after $\overline{RDY}$ has been driven low. $\overline{ADS}$ must return to an inactive high state before the next cycle can be issued. It is highly recommended not to have, $\overline{ADS}$ more than one clock wide.

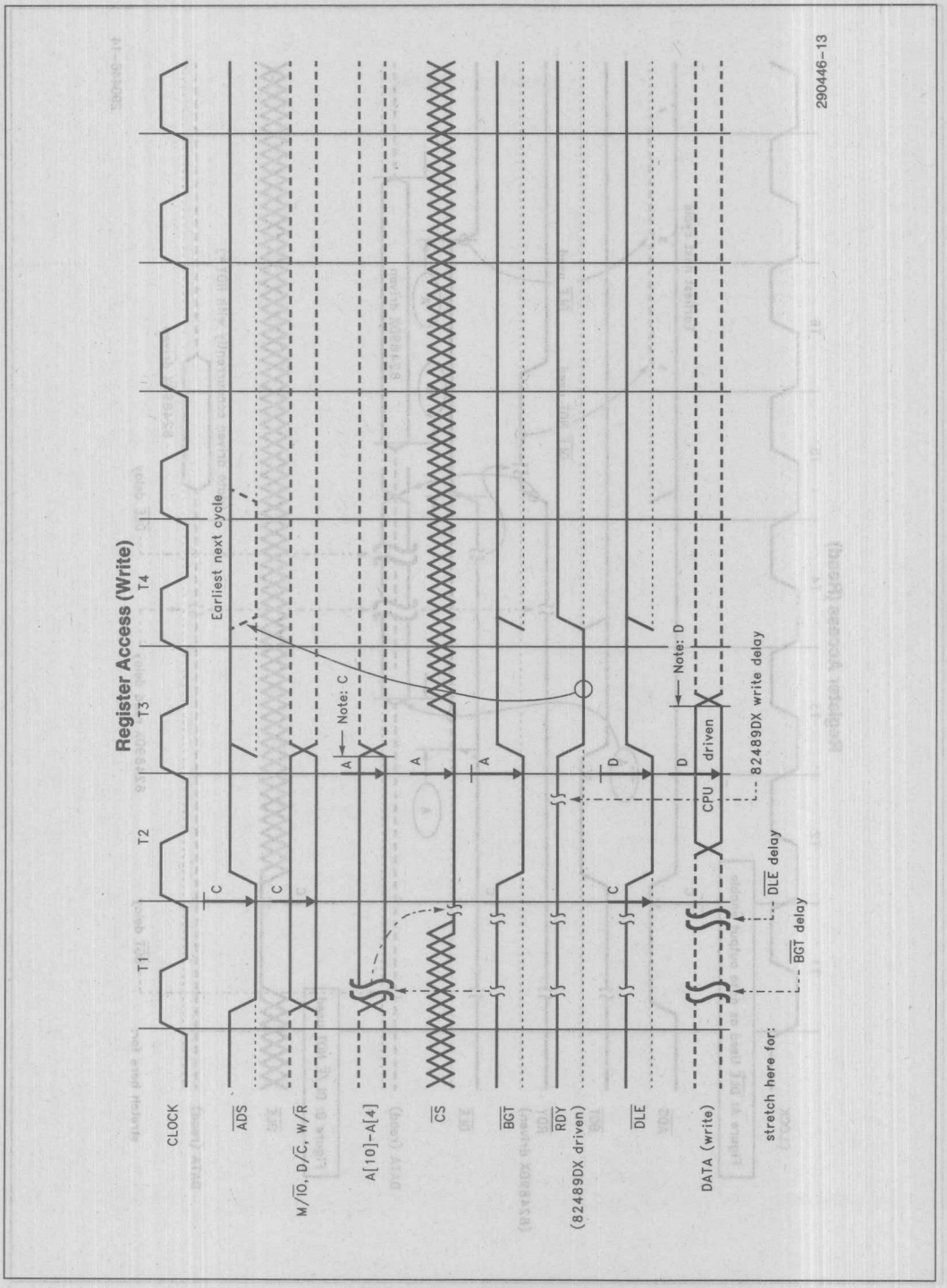


Figure 23. Timing Diagram 1.

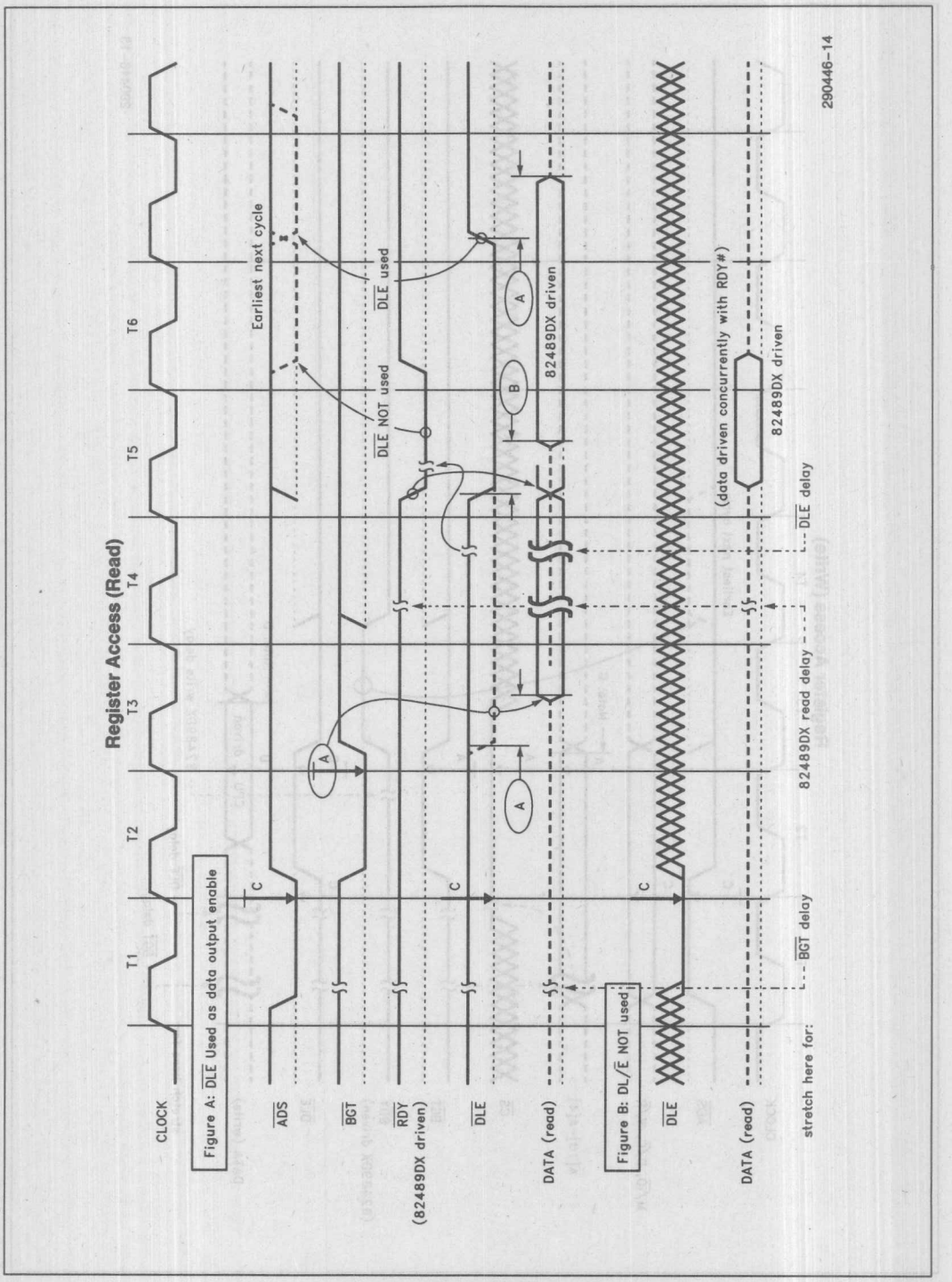


Figure 24. Timing Diagram 2

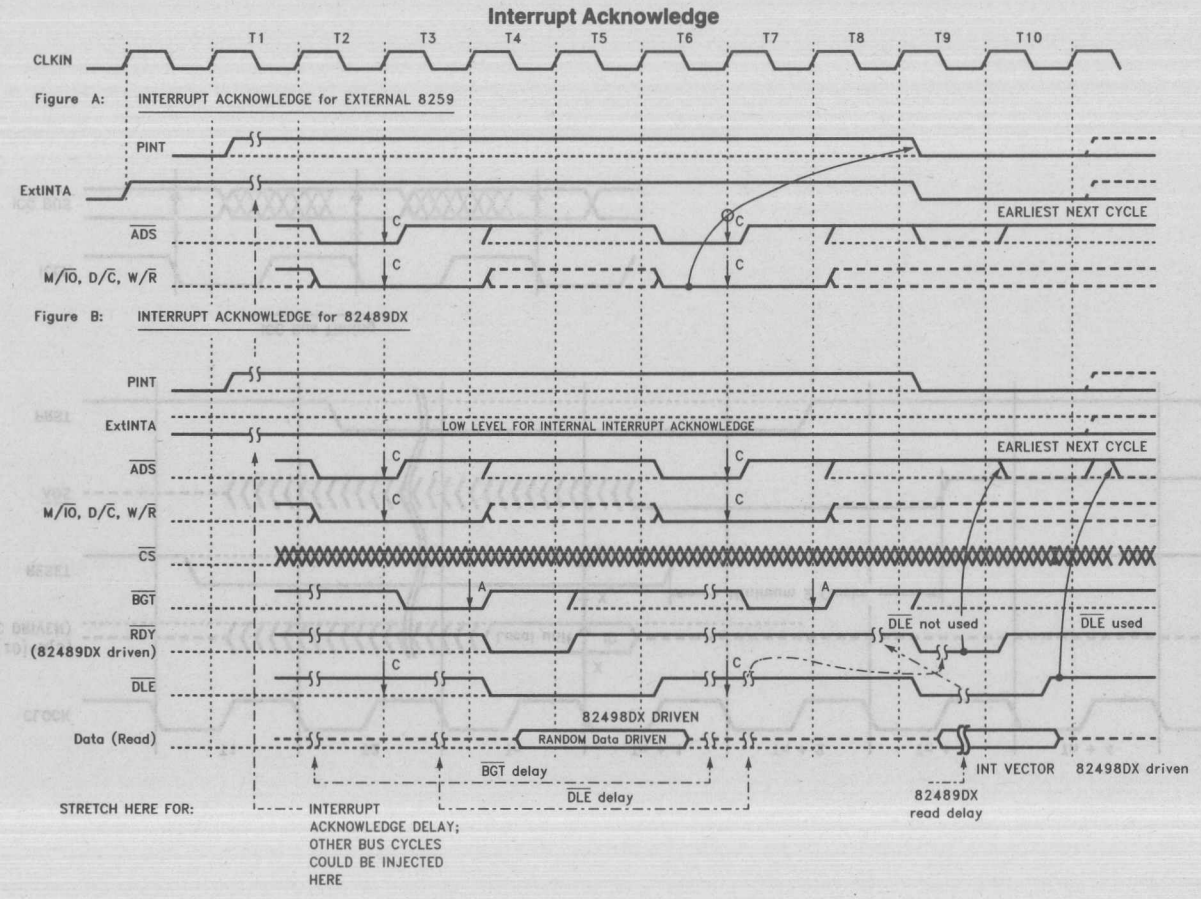


Figure 25. Timing Diagram 3

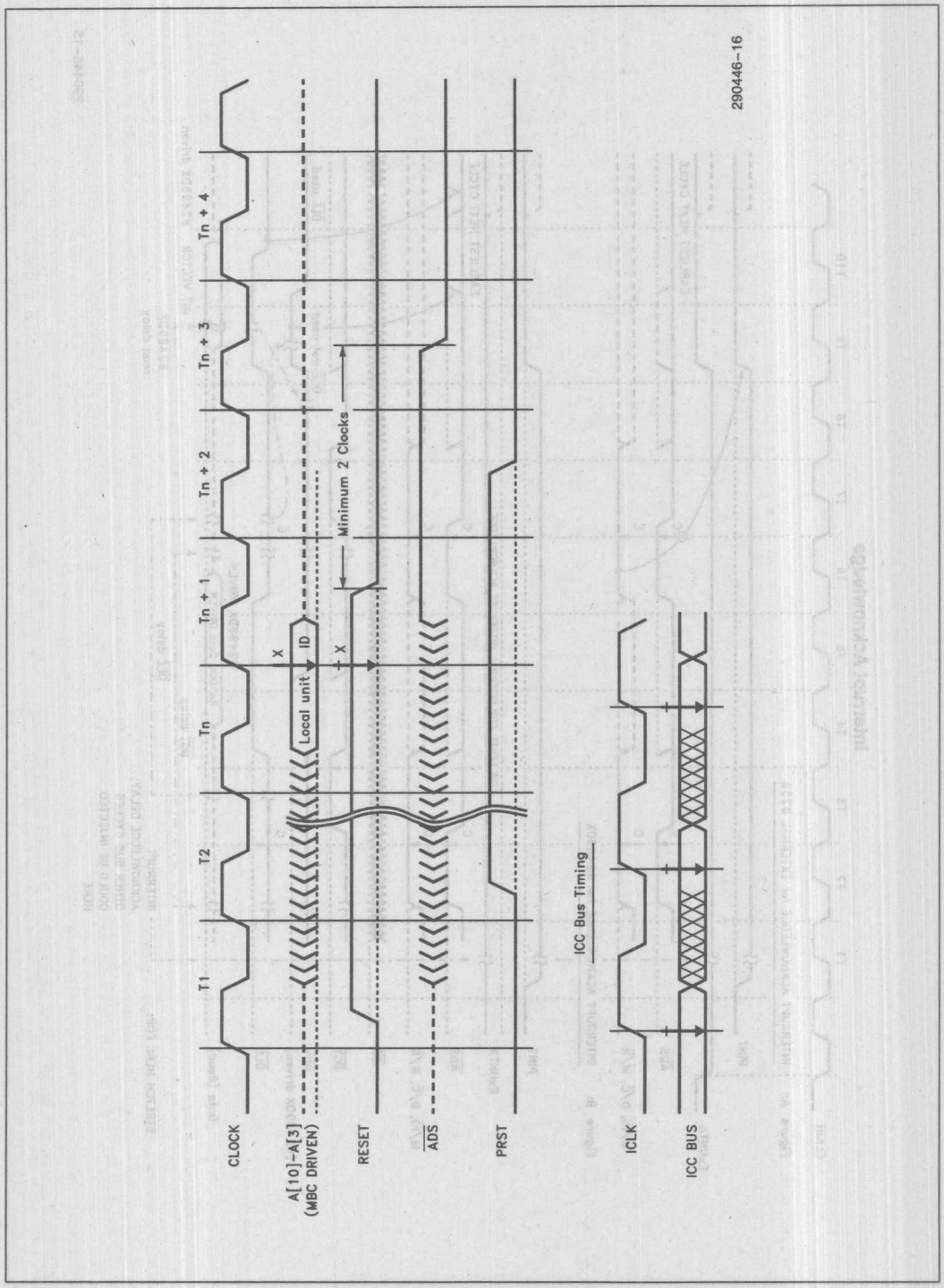


Figure 26. Timing Diagram 4

REGISTER READ TIMING

For discussion of this bus cycle, refer to Figure 24, 82489DX timing Diagram 2. It shows the relationship of the three phases of the bus cycle, however, the dependent control and address signals are not shown here, since they behave exactly as in the case of a register write. See the previous section for the description of control and address phases of the bus cycle.

In the case of a read when $\overline{DLE}$ is used (Figure 24A), it works logically like an asynchronous, output data enable. The 82489DX drives the data bus within time delay "A" after $\overline{DLE}$ is asserted, which must not occur before $\overline{BGT}$. Note that even though the bus is being driven, the data only becomes valid during the clock cycle in which $\overline{RDY}$ is asserted, after that point, valid data is maintained on the bus as long as $\overline{DLE}$ remains asserted, after which the data bus returns to high impedance state within time delay "B" of $\overline{DLE}$ deassertion. If $\overline{DLE}$ is asserted late, 82489DX could complete its internal read cycle and return $\overline{RDY}$ early. In that case, $\overline{RDY}$ active low state will be maintained until $\overline{DLE}$ is asserted.

In the case of a read when $\overline{DLE}$ is NOT used (Figure 24B) the data is driven for exactly one clock cycle, coincident with $\overline{RDY}$ being asserted.

$\overline{DLE}$ is sampled with the control signals to determine whether it is being used. If sampled in the asserted state when $\overline{ADS}$ is active, the $\overline{DLE}$ will be considered not used, and its state during the remainder of the cycle doesn't matter. This is consistent with the notion of permanently tying this signal low when not used, as described in the previous section.

Indication of the end of the bus cycle is dependent on the use of $\overline{DLE}$. When it is not used, (i.e., permanently tied to ground) $\overline{RDY}$ indicates the end of the bus cycle, as it does in the case of write access. When it is used, $\overline{DLE}$ deassertion indicates the end of the cycle, since the 82489DX could be driving the bus well after $\overline{RDY}$ is deasserted. In either case, the 82489DX can accept the next $\overline{ADS}$ pulse anytime after $\overline{RDY}$ has been asserted. However, note that if $\overline{DLE}$ is being used, the next $\overline{ADS}$ should be delayed until $\overline{DLE}$ can be safely sampled inactive. Note these two options for earliest next cycle in the timing diagram.

INTERRUPT ACKNOWLEDGE TIMING

For discussion of this bus cycle, refer to Figure 25 82489DX timing diagram 3, This cycle is the result of the 82489DX posting an interrupt to the processor by asserting PINT. After PINT is asserted, other bus cycles may occur before the interrupt acknowledge cycle.

PINT can be asserted for any external (8259) interrupt as shown in Figure 25A, or for an 82489DX generated interrupt as shown in Figure 25B. ExtINTA indicates whether PINT was asserted in response to an 8259 request or an 82489DX request. This signal is used by external control logic to either allow or preclude the 8259 from responding to the subsequent interrupt acknowledge cycle. It should be noted that ExtINTA pin gets deactivated at third clock after the $\overline{ADS}$ of the second INTA cycle.

When ExtINTA is high, the 82489DX will not respond to the acknowledge cycle other than to deassert PINT signal, and clear the pending external interrupt two full clock cycles after the $\overline{ADS}$ of the second INTA cycle, as shown in Figure 25A. Once PINT is asserted in response to an external interrupt, it can only be deasserted by an INTA cycle. An INTA cycle is recognized by 82489DX as soon as the bus cycle definition is sampled with $\overline{ADS}$ in a low state. $\overline{BGT}$ is not needed in this case.

When ExtINTA is low, the 82489DX will respond to the acknowledge cycle, as shown in Figure 25B. In this case, external logic (e.g., the memory bus controller) is expected to prevent any attached 8259 from seeing the acknowledge cycle. When PINT is asserted in response to an 82489DX internal interrupt, it can be deasserted by an INTA cycle. The PINT signal will be deasserted 5 full clocks after $\overline{BGT}$ of the second INTA cycle.

Note that ExtINTA is stable at all times while PINT is asserted. That means that even if new interrupts arrive between the time an interrupt is posted to the processor, and the acknowledge occurs, the 82489DX will not change its commitment for an external (8259) or internal (82489DX) acknowledge cycle, regardless of priority. This also means that PINT may be raised for a high priority internal interrupt right after responding to the external Interrupt. In any event, PINT will be kept low for a minimum of two clocks before reasserting itself.

The interrupt acknowledge cycle is indicated by the bus cycle definition signals all being low, and looks like two consecutive read cycles, except that there is no explicit address information. The actual content of the address pins during this cycle is processor dependent, and therefore there is no chip select either. Chip select is implied by a combination of the bus cycle definition signals (all low) and $\overline{BGT}$.

Note that there is a "dummy" data phase in the first interrupt acknowledge cycle. This allows parity to be generated on the bus for processors like i860XP. During this cycle, 82489DX drives random data on the bus with the appropriate parity. The interrupt Request Register is "frozen" and the highest priority

pending interrupt vector is returned to the processor in the second acknowledge cycle.

The second acknowledge cycle has a complete data phase with timings identical to those of an ordinary register read. The data returned is the vector of the highest priority internally pending 82489DX interrupt, or the spurious interrupt vector, if there is no interrupt pending higher than the current processor priority.

Note that the timing diagram shows $\overline{DLE}$ being used (sampled high during $\overline{ADS}$). However, just as in a normal read cycle, the option exists not to use $\overline{DLE}$ (i.e., permanently tied to ground).

RESET AND MISCELLANEOUS TIMING

For discussion of this bus cycle, refer to Figure 26 82489DX Timing Diagram 4. It shows the 82489DX reset cycle, the timing of some related signals, and the ICC bus. The RESET input has a setup and hold time to the system clock edge, CLKIN, as do other independently timed signals, The RESET signal will reset the two asynchronous system on the chip, namely the ICC bus unit running synchronously to the ICLK and all the other unit running synchronous to the system clock, CLKIN. RESET must meet the minimum reset time with respect to both clocks and there should be at least one ICLK rising edge during reset. The TAP controller should also be initialized.

During reset, an eight-bit 82489DX Local Unit ID can be optionally initialized. Eight address lines, A10–A3 are sampled on every clock edge while RESET is asserted. The last sample remains in the 82489DX Local Unit ID register after reset. Alternatively, the 82489DX Local Unit ID can be loaded with a register write as part of software initialization, before 82489DX operation is started. In any event, the register must be initialized before the 82489DX can communicate on the ICC bus, including sending/receiving RESET messages. All valid signal to 82489DX should wait at least two full clocks after RESET is deasserted.

The PRST signal (reset output) is asserted both with RESET input, or under software control. Its on-off delay times are relative to the rising clock edge. The duration of PRST under software control is defined by the software itself. Also note that the PNMI pin has the same timing as does PRST when the latter is software controlled.

The ICC bus signals are both input and output on each cycle. Setup, hold and delay times are all measured with respect to the ICC bus Clock ICLK which has no relationship to the Processor clock on which the remainder of the 82489DX runs. This means that the ICC bus is independently sampled on each ICLK edge, as shown. It also implies that largest possible hold time will not exceed the minimum delay time.

After reset, all 82489DX registers are reset to "0" state. The mask bits in the local vector table and the redirection table are reset to "1" state to mask out all interrupts. All reserved bits are all wired to "0" state permanently on chip.

10.0 BOUNDARY SCAN DESCRIPTION

The 82489DX is equipped with the JTAG boundary scan standard. This feature allows the user to test the interconnections between 82489DX and the external hardware once they have been assembled onto a printed circuit board or other substrate. In addition to the JTAG mandatory instruction set, 82489DX also provides the INTEST instruction which allows static testing of the on-chip logic.

The detailed information related to the IEEE Std 1149.1-1990 (the JTAG standard) can be obtained from the reference document *IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std 1149.1–1990)*.

10.1 Boundary Scan Architecture

The boundary scan logic contains the following elements:

- **Five Test Access Ports (TAP):** They are labeled as $\overline{trst}$, tck, tdi, tdo and tms. All ports are input pins except tdo, which is a tri-state output pin.
- **A TAP Controller:** The logic is used to control the boundary scan activity.
- **82489DX Device ID Register:** This is a 32-bit read-only register. The DID can be shifted out in ascending order to the tdo pin.
- **JTAG Instruction Register (IR):** This is a 4-bit register which accepts instruction code shifted in from the tdi pin. The opcode stored in the IR register is used to control operation.
- **Boundary Scan Register:** This is a 137 stages scan path which connects almost all 82489DX signal pins for boundary scan purposes.
- **Bypass Register:** This register simply allows the data which goes into tdi pin to be shifted out directly from tdo.

The following block diagram illustrates the implementation of the JTAG architecture in the 82489DX design.

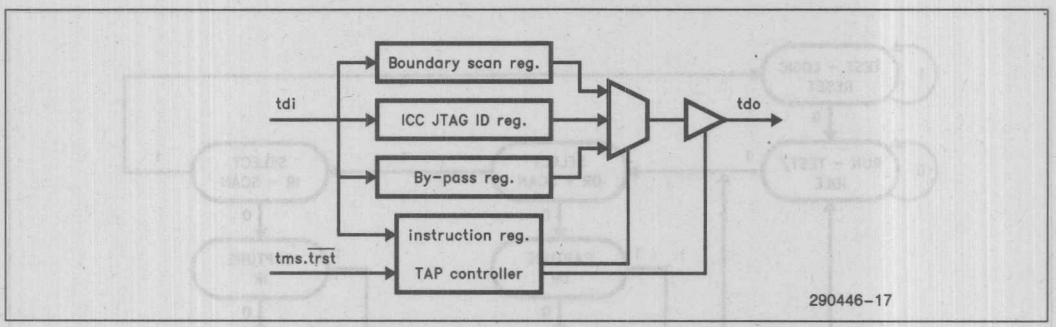


Figure 27. Block Diagram of the JTAG Architecture

Test Access Ports

$\overline{trst}$ TAP controller master reset pin. When $\overline{trst}$ is low, the TAP controller's state machine will be reset to "test-logic-reset" state asynchronously. This pin is tied to a weak internal pull-up for keeping to be a logical 1 when not driven.

tck This is the test logic clock. The test logic will change state on the rising edge of the tck.

tdi Test data input. Data is shifted into the tdi pin on the rising edge of tck. This pin is tied to a weak internal pull-up for keeping it to be a logical 1 when not driven.

tms Test mode select. This pin is used to select the state of the TAP controller. This pin is synchronous to the rising edge of the tck. This pin is tied to a weak internal pull-up for keeping it to be a logical 1 when not driven.

tdo Test data output. This is a tri-state pin which allows the data to be shifted out.

TAP CONTROLLER

The TAP controller in 82489DX is implemented to conform the IEEE1149.1 standard. The TAP controller is a single phase clock, synchronous finite state machine. It controls the sequence of the operation of the test logic.

The value of the test mode state (tms) pin at a rising edge of tck controls the sequence of the state changes. The state diagram for the TAP controller is shown in Figure 28. Test designers must consider the operation of the state machine in order to have the correct sequence of value to drive on tms.

The behavior of the TAP controller and other test logic in each of the controller states is briefly described as follows.

4

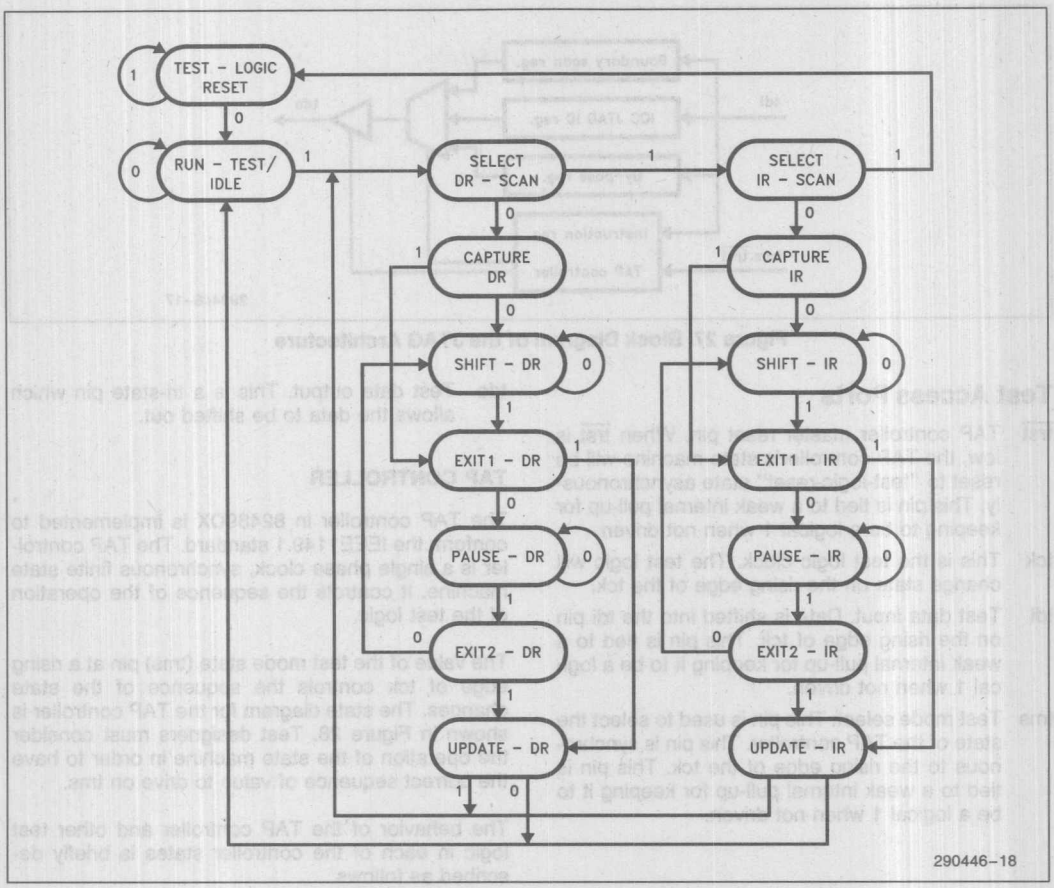


Figure 28. TAP Controller State Diagram

TEST-LOGIC-RESET

The test logic is disabled so that normal operation of the on-chip system logic (i.e., in response to stimuli received through the system pins only) can continue unhindered. This is achieved by initializing the instruction register to contain the IDCODE instruction. No matter what the original state of the controller, it will enter Test-Logic-Reset when tms is held high for at least five rising edges of tck. The controller remains in this state while tms is high.

If the controller should leave the Test-Logic-Reset controller state as a result of an erroneous low signal on the tms line at the time of the rising edge on tck (for example, a glitch due to external interference), it will return to the Test-Logic-Reset state following three rising edges of tck with the tms line at the intended high logic level. The operation of the test logic is such that no disturbance is caused to on-chip system logic operation as the results of such an error. On leaving the Test-Logic-Reset controller state, the controller moves into the Run-Test/Idle controller state where no action will occur because the current instruction has been set to select operation of the device identification register. The test logic is also inactive in the Select-DR-Scan and Select-IR-Scan controller states.

Note that the TAP controller will also be forced to the Test-Logic-Reset controller state asynchronously by applying a low logic level at $\overline{trst}$.

RUN-TEST/IDLE

A controller state between scan operations. Once entered, the controller will remain in the Run-Test/Idle state as long as tms is held low. When tms is high and a rising edge is applied at tck, the controller moves to the Select-DR-Scan state.

In the Run-Test/Idle controller state, activity in selected test logic occurs only when certain instructions are present such as RUNBIST. Since 82489DX does not have RUNBIST instruction, this state is acting like an idle state.

The instruction does not change while the TAP controller is in this state.

SELECT-DR-SCAN

This is a temporary controller state in which all test data register (82489DX has one test data register which is the boundary scan shift registers path) selected by the current instruction retain their previous state.

If tms is held low and a rising edge is applied to tck when the controller is in this state, then the controller moves into the Capture-DR state and a scan sequence for the selected test data register is initiated. If tms is held high and a rising edge is applied to tck, the controller moves on to the Select-IR-Scan state.

The instruction does not change while the TAP controller is in this state.

SELECT-IR-SCAN

This is a temporary controller state in which all test data registers selected by the current instructing retain their previous state.

If tms is held low and a rising edge is applied to tck when the controller is in this state, then the controller moves into the Capture-IR state and a scan sequence for the instruction register is initiated. If tms is held high and a rising edge is applied to tck, the controller returns to the Test-Logic-Reset state. **The instruction does not change while the TAP controller is in this state**

CAPTURE-DR

In this controller state data may be parallel-loaded into test data registers selected by the current instruction on the rising edge of tck. If a test data register selected by the current instruction does not have a parallel input, or if capturing is not required for the selected test, then the register retains its previous state unchanged. ***The instruction does not change while the TAP controller is in this state.***

When the TAP controller is in this state and a rising edge is applied to tck, the controller enters either the Exit 1-DR state if tms is held at 1 or the Shift-DR state if tms is held at 0.

SHIFT-DR

In this controller state, the test data register connected between tdi and tdo as a result of the current instruction shifts data one stage towards its serial output on each rising edge of tck. Test data registers that are selected by the current instruction, but are not placed in the serial path, retain their previous state unchanged. ***The instruction does not change while the TAP controller is in this state.***

When the TAP controller is in this state and a rising edge is applied to tck, the controller enters either the Exit 1-DR state if tms is held at 1 or remains in the Shift-DR state if tms is held at 0.

EXIT 1-DR

This is a temporary controller state, if tms is held high, a rising edge applied to tck while in this state causes the controller to enter the Update-DR state, which terminates the scanning process. If tms is held low and a rising edge is applied to tck, the controller enters the Pause-DR state. **All test date registers selected by the current instructions retain their previous state unchanged.**

The instruction does not change while the TAP controller is in this state.

PAUSE-DR

This controller state allows shifting of the test data register in the serial path between tdi and tdo to be temporarily halted. All test data registers selected by the current instruction retain their previous state unchanged.

The controller remains in this state while tms is low. When tms goes high and a rising edge is applied to tck, the controller moves on to the Exit 2-DR state. The instruction does not change while the TAP controller is in this state.

EXIT 2-DR

This is a temporary controller state. If tms is held high and a rising edge is applied to tck while in this state, the scanning process terminates and the TAP controller enters the Update-DR controller state. If tms is held low and a rising edge is applied to tck, the controller enters the Shift-DR state.

All test data registers selected by the current instruction retain their previous state unchanged. The instruction does not change while the TAP controller is in this state.

UPDATE-DR

Some test date registers may be provided with a latched parallel output to prevent changes at the parallel output while data is shifted in the associated shift-register path in response to certain instructions (e.g., EXTENT, INTEST, and RUNBIST). Data is latched onto the parallel output of these test data registers from the shift-register path on the falling edge of tck in the Update-DR controller state. The data held at the latched parallel output should not change other than in this controller state unless operation during the execution of a self test is required (e.g., during the Run-Test/Idle controller state in response to a design-specific public instruction).

All shift-register stages in test data registers selected by the current instruction retain their previous state unchanged. **The instruction does not change while the TAP controller is in this state.**

When the TAP controller is in this state and a rising edge is applied to tck, the controller enters either the Select-DR-Scan state if tms is held at 1 or the Run-Test/Idle state if tms is held at 0.

CAPTURE-IR

In this controller state the shift-register contained in the instruction register loads a pattern of fixed logic values on the rising edge of tck. In addition, design-specific data may be loaded into shift-register stages that are not required to be set to fixed values.

Test data registers selected by the current instruction retain their previous state. The instruction does not change while the TAP controller is in this state.

When the TAP controller is in this state and rising edge is applied to tck, the controller enters either the Exit 1-IR state tms is held at 1 or the Shift-IR state if tms is held at 0.

SHIFT-IR

In this controller state the shift-register contained in the instruction register is connected between tdi and tdo and shifts data one stage towards its serial output on each rising edge of tck.

Test data registers selected by the current instruction retain their previous state. The instruction does not change while the TAP controller is in this state. When the TAP controller is in this state and a rising edge is applied to tck, the controller enters either the Exit1-IR state if tms is held at 1 or remains in Shift-IR state if tms is held at 0.

EXIT 1-IR

This is a temporary controller state. If tms is held high, a rising edge applied to tck while in this state causes the controller to enter the Update-IR state, which terminates the scanning process. If tms is held low and a rising edge is applied to tck, the controller enters the Pause-IR state.

Test data registers selected by the current instruction retain their previous state. The instruction does not changes while the TAP controller is in this state and the instruction register retains its state.

PAUSE-IR

This controller state allows shifting of the instruction register to be halted temporarily.

Test data registers selected by the current instruction retain their previous state. The instruction does not change while the TAP controller is in this state and the instruction register retains its state.

The controller remains in this state while tms is low. When tms goes high and a rising edge is applied to tck, the controller moves on to the Exit 2-IR state.

EXIT 2-IR

This is a temporary controller state. If tms is held high and a rising edge is applied to tck while in this state, termination of the scanning process results, and the TAP controller enters the Update-IR controller state. If tms is held low and a rising edge is applied to tck, the controller enters the Shift-IR state.

Test data registers selected by the current instruction retain their previous state. The instruction does not change while the TAP controller is in this state and the instruction register retains its state.

UPDATE-IR

The instruction shifted into the instruction register is latched onto the parallel output from the shift-register path on the falling edge of tck in this controller state. Once the new instruction has been latched, it

becomes the current instruction. **Test data registers selected by the current instruction retain their previous state.**

When the TAP controller is in this state and a rising edge is applied to tck, the controller enters the Select-DR-Scan states if tms is held at 1 or the Run-Test/Idle state if tms is held at 0.

INSTRUCTION REGISTER

The function of the instruction register is to select the operating mode of the test logic. For instance, read the ID register, or capture the 82489DX output signals. 82489DX has implemented 4 instructions.

Instruction	Mandatory/Optional	Opcode
bypass	m	1111
extest	m	0000
sample/preload	m	0001
idcode	m	0010
reserved	o	1001

Bypass Instruction

The bypass instruction selects the bypass register to be connected to tdi and tdo, effectively bypassing the test logic on the 82489DX boundary scan path and reducing the shift length to be on one bit. Note that an open circuit fault in the board level test data path will cause the bypass register to be selected following an instruction scan cycle due to the internal pull-up on the tdi pin. This has been done to prevent any unwanted interference with the proper operation of the system logic.

Extest Instruction

The extest instruction allows testing of circuitry external to the component package, typically board interconnects. It does so by driving the values loaded into the 82489DX's boundary scan register out on the output pins corresponding to each boundary scan cell and capturing the values on 82489DX's input pins to be loaded into their corresponding boundary scan register locations. I/O pins are selected as input or output depending on the value located into the output control cell. Values shifted into input latch in the boundary scan register are never used by the internal logic of the 82489DX.

NOTE:
82489DX must be reset after extest instruction has been executed.

Sample/Preload Instruction

The sample/preload instruction has two functions that it can perform. When the TAP controller is in the CAPTURE-DR state, the sample/preload instruction allows a snap-shot of the normal operation of the 82489DX without interfering with that normal operation. The instruction causes boundary scan register cells associated with outputs to sample the value being driven into the 82489DX. On both outputs and inputs the sampling occurs on the rising edge of tck. When preloads data into the 82489DX pins to be driven to the board by executing the extest instruction. Data is preloaded to the pins from the boundary scan register on the falling edge of tck.

dcode Instruction

The idcode instruction selects the device identification register to be connected to tdi and tdo, allowing the device ID code to be shifted out of the device on tdo. Note that the bit stream shifted into tdi will appear on tdo after all 32 bits of the DID has been shifted out.

4

DEVICE IDENTIFICATION REGISTER (DID)

The device identification is a 32 bits number which can be read by the external hardware by using the idcode instruction. The 82489DX device ID is assigned to 1489A013 (hex). This is subject to change. The upper 4 bits of DID may be changed for different version. The 16-bit number (bit 27–bit 12) 489A (hex) is the part ID. The lower 12 bits are the manufacturer ID for Intel which must be 013 (hex).

BOUNDARY SCAN REGISTER

82489DX has only one test data register, i.e., the boundary scan register. The boundary scan register is a single shift register path containing the boundary scan cells that are connected to all signal input and output pins of the 82489DX. There are three generic type of boundary scan cells—input, output, and bi-directional. For each input only cell, one stage of shift register is added to the boundary scan path.

All output pins will become tri-stateable when boundary scan is activated, regardless whether they are tri-stateable or not in the normal operation. To explain further, the user will enable/disable an output driver with a specific tri-state control cell in the scan path. The user must shift in a proper control signal for these tri-state control cells in the scan path.

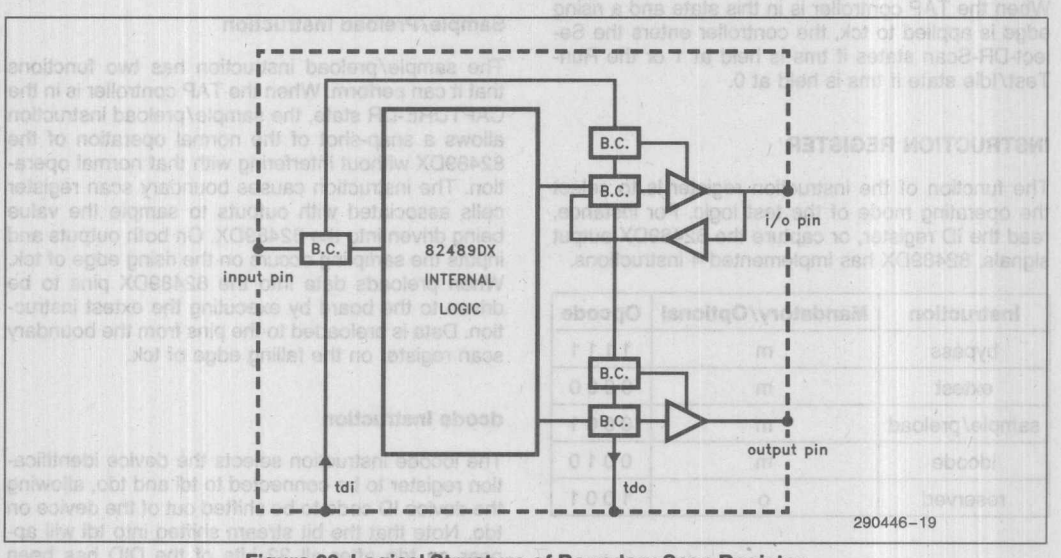


Figure 29. Logical Structure of Boundary Scan Register

BOUNDARY SCAN CELL NAMES IN ORDER FROM tdi TO tdo

The following table is a list of the boundary scan cell names in the order from tdi to tdo. The type information indicates the purpose of the cells.

I = input only cell
B = bi-directional cell
T = tri-state output cell
C = tri-state control. Note that the signal name enclosed within the parenthesis is controlled by this cell.

Cell Number	Type	Name	Pin #
1	I	CLKIN	57
2	I	TMBASE	59
3	I	ICLK	60
4	I	$\overline{DC}$	61
5	I	$\overline{WR}$	62
6	I	M/$\overline{IO}$	63
7	I	$\overline{ADS}$	64
8	I	RESET	65
9	I	$\overline{BGT}$	66
10	I	reserved	70
11	I	reserved	71
12	I	reserved	72
13	I	$\overline{DLE}$	73
14	I	$\overline{CS}$	74
15	I	reserved	75
16	I	MBI3	76
17	I	MBI2	77
18	I	MBI1	78
19	I	MBI0	79
20	I	LINTIN1	80
21	I	LINTIN0	81
22	T	reserved	
23	C	(reserved)	
24	I	INTIN15	82
25	I	INTIN14	83
26	T	reserved	
27	C	(reserved)	
28	I	INTIN13	84
29	I	INTIN12	85
30	T	reserved	
31	C	(reserved)	
32	I	INTIN11	86
33	I	INTIN10	87
34	T	reserved	
35	C	(reserved)	
36	I	INTIN9	88
37	I	INTIN8	89
38	T	reserved	
39	C	(reserved)	
40	I	INTIN7	90
41	I	INTIN6	91
42	T	reserved	
43	C	(reserved)	
44	I	INTIN5	92
45	I	INTIN4	93
46	T	reserved	
47	C	(reserved)	
48	I	INTIN3	94
49	I	INTIN2	95
50	T	reserved	
51	C	(reserved)	
52	I	INTIN1	96
53	I	INTIN0	97
54	I	reserved	
55	B	DP3	101
56	C	(DP[3:0])	
57	B	DP2	102
58	B	DP1	103
59	B	DP0	104
60	T	reserved	
61	C	(reserved)	
62	B	D31	105
63	C	(D[31:0])	
64	B	D30	107

Cell Number	Type	Name	Pin #
65	B	D29	109
66	B	D28	110
67	T	reserved	
66	C	(reserved)	
69	B	D27	111
70	B	D26	112
71	T	reserved	
72	C	(reserved)	
73	B	D25	114
74	B	D24	115
75	B	D23	116
76	T	reserved	
77	C	(reserved)	
78	B	D22	118
79	B	D21	119
80	B	D20	121
81	B	D19	122
82	B	D18	123
83	B	D17	124
84	B	D16	125
85	B	D15	128
86	B	D14	129
87	B	D13	130
88	B	D12	131
89	B	D11	2
90	T	reserved	
91	C	(reserved)	
92	B	D10	3
93	B	D9	4
94	T	reserved	
95	C	(reserved)	
96	B	D8	7
97	B	D7	8
98	B	D6	9
99	B	D5	11
100	B	D4	12
101	B	D3	13

Cell Number	Type	Name	Pin #
102	B	D2	14
103	B	D1	16
104	B	D0	18
105	I	reserved	19
106	B	reserved	20
107	C	(cell106, A[10:3])	
108	B	A10	21
109	B	A9	22
110	B	A8	24
111	B	A7	26
112	B	A6	27
113	B	A5	28
114	B	A4	29
115	B	A3	31
116	T	reserved	34
117	C	reserved	
118	T	PINT	35
119	C	(PINT)	
120	T	PNMI	37
121	C	(PNMI)	
122	T	PRST	38
123	C	(PRST)	
124	T	ExtINTA	41
125	C	(reserved)	
126	T	reserved	42
127	C	(reserved)	
128	T	$\overline{\text{RDY}}$	43
129	C	($\overline{\text{RDY}}$)	
130	T	MBO3	45
131	C	(MBO3)	
132	T	MBO2	48
133	C	(MBO2)	
134	T	MBO1	49
135	C	(MBO1)	
136	T	MBO0	51
137	C	(MBO0)	

BYPASS REGISTER

The bypass register is simply a 1-bit shift register which connects between the tdi and tdo. When selected by using the bypass instruction, the data shifted into tdi will be shifted out from tdo one tck clock later.

JTAG TAP Controller Initialization

The TAP controller must be reset to test-logic-reset state when 82489DX is first powered up. There are two ways to reset the TAP controller:

1. Assert $\overline{\text{trst}}$ to be 0, it will reset the TAP controller asynchronously.
2. Assert tms to be 1, and clock the TAP controller at least five times, the TAP controller will be reset after the fifth rising edge of the tck.

After reset, the idcode instruction is loaded into the IR automatically.

Note that the tms and $\overline{\text{trst}}$ pins both have an internal weak pull-up device to keep them to be logic 1 level. Therefore the user can simply apply 5 clocks at the tck input to reset the TAP controller. If the TAP controller is not reset properly, 82489DX may not function because the boundary scan logic might be active which will impact the signals flow in and out to the chip.

11.0 ELECTRICAL CHARACTERISTICS

11.1 D.C. Specifications

NOTICE: This data sheet contains information on products in the sampling and initial production phases of development. The specifications are subject to change without notice. Verify with your local Intel Sales office that you have the latest data sheet before finalizing a design.

ABSOLUTE MAXIMUM RATINGS

Case Temperature Under Bias . . . −65°C to +110°C

Storage Temperature . . . . . −65°C to +150°C

Voltage on Any Pin with Respect to Ground . . . . . −0.5 to V_{CC} + 0.5

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

V_{CC} = 5V ±5%; T_C = 0°C to +85°C

Symbol	Parameter	Min (ns)	Max	Units	Notes
V_{IL}	Input LOW Voltage (TTL)	−0.3	+0.8	V	
V_{IH}	Input HIGH Voltage (TTL)	2.0	V_{CC} + 0.3	V	
V_{OL}	Output LOW Voltage (TTL)		+0.45	V	(Note 1)
V_{QH}	Output HIGH Voltage (TTL)	2.4		V	(Note 2)
I_{CC}	33 MHz Power Supply Current		200	mA	
I_{LI}	Input Leakage Current		15	μA	
I_{LL}	Input Leakage Current		−600	μA	(Note 5)
I_{LH}	Output Leakage Current		600	μA	(Note 4)
I_{LO}	Output Leakage Current		15	μA	(Note 3)
C_{IN}	Input Capacitance		3	pF	
C_O	I/O or Output Capacitance		6	pF	
C_{CLKIN}	Clock Capacitance		3	pF	
I_{MLO}	ICC Bus Output Low Current		4	mA	(Note 6)
C_{MC}	ICC Bus Total Capacitance		100	pF	
V_{MH}	ICC Bus Input High (TTL)	2.0	V_{CC} + 0.3	V	
V_{ML}	ICC Bus Input Low (TTL)	−0.3	+0.8	V	

NOTES:
1. This parameter is measured with current load of 4 mA.
2. This parameter is measured with current load of 1.0 mA.
3. This parameter is for output without pulldown.
4. This parameter is for tri-state output with pulldown and V_{OH} = 3.0V.
5. This parameter is for input with pullup at V_{IL} = 0V.
6. ICC bus output low current is measured at 0.6V.

11.2 A.C. Specifications

A.C. Parameters Referencing 33 MHz System Clock

V_{CC} = 5V ±5%; T_C = 0°C to +85°C

Symbol	Parameter	Ref. Fig.	Load (pF)	Min (ns)	Max (ns)	Notes
tc	CLKIN Period	30		30	100	(Note 1)
t1	CLKIN High Time	30		5		
t2	CLKIN Low Time	30		5		
t3	CLKIN Rise Time	30			3	(Note 2)
t4	CLKIN Fall Time	30			3	(Note 2)
t5	$\overline{ADS}$, $\overline{BGT}$, $\overline{DLE}$, M/$\overline{IO}$, D/$\overline{C}$, W/$\overline{R}$, $\overline{CS}$ Setup Time	31		8		
t6	D31–D0, DP3–DP0, A9–A3 Setup Time	31		8		
t8	$\overline{ADS}$, $\overline{BGT}$, $\overline{DLE}$, M/$\overline{IO}$, D/$\overline{C}$, W/$\overline{R}$, $\overline{CS}$ Hold Time	31		5		
t10	D31–D0, DP3–DP0, A9–A3 Hold Time	31		5		
t11	D31–D0, DP3–DP0, Valid Delay	30	50		18	
t12	D31–D0, DP3–DP0, Low-Z Delay When $\overline{DLE}$ is Not Used	32	50	3		(Note 7)
t13	D31–D0, DP3–DP0, High-Z Delay When $\overline{DLE}$ is Not Used	32	50		14	(Note 7)
t14	D31–D0, DP3–DP0 Enable Delay When $\overline{DLE}$ is Used	33	50	3	42	
t15	D31–D0, DP3–DP0 Disable Delay When $\overline{DLE}$ is Used	33	50	3	14	
t20	$\overline{RDY}$ Valid Delay	30	50	3	18	
t21	PRST, PNMI, PINT Valid Delay	30	50	3	34	
t22	RESET Setup Time	31		8		(Note 5)
t23	RESET Hold Time	31		5		(Note 5)
	RESET Cycle Time			5 tc		(Note 3)
				1 tic		(Note 3)
t24	INTIN[15:0], LINTIN[1:0] Low Time			10		(Note 6)

All parameters are given in nanoseconds.
TTL Level timing is measured at 1.5V for both "0" and "1" levels.

NOTES:
1. ICC bus clock ICLK period must be at least 5 ns longer than system clock CLKIN for proper synchronization of the internal asynchronous signals.
2. System clock CLKIN measured from 0.8V–2.0V.
3. Minimum Reset cycle is the greater of the two cycle times.
4. Minimum pulse width must be met for valid level to be attained on the DATA or ADDRESS output.
5. Set up and hold time is required for RESET to start at the next rising edge of the clock.
6. INTIN and LINTIN low time is measured from 1.5V of the falling edge to 1.5V of rising edge.
7. Not 100% tested. Guaranteed by design characterization.

Time Base A.C. Parameters

V_{CC} = 5V ±5%; T_C = 0°C to +85°C

Symbol	Parameter	Ref. Fig.	Min (ns)	Max (ns)	Note
tmc	TMBASE Period	35	40	10000	
t30	TMBASE High Time	35	10		
t31	TMBASE Low Time	35	10		
t32	TMBASE Rise Time	35		8	
t33	TMBASE Fall Time	35		8	

TAP Controller A.C. Parameters V_{CC} = 5V ±5%; T_C = 0°C to +85°C

Symbol	Parameter	Ref. Fig.	Min (ns)	Max (ns)	Note
ttc	TCK Period	35	40	1000	
t50	TCK High Time	35	10		
t51	TCK Low Time	35	10		
t52	TCK Rise Time	35		8	
t53	TCK Fall Time	35		8	
t54	TDI, TMS, $\overline{\text{TRST}}$ Setup Time	34	10		
t55	TDI, TMS, $\overline{\text{TRST}}$ Hold Time	34	5		
t56	TDO VALID Delay	34	5	24	(Note 1)
t57	Output Delay in EXTest in EXTEST Mode	34	5	27	(Note 1)
t58	$\overline{\text{TRST}}$ Minimum Low Time		10		(Note 2)

All parameters are given in nanoseconds.
TTL level timing is measured at 1.5V for both "0" and "1" levels.

NOTES:
1. These parameters are specified for 50 pF load.
2. This parameter is measured at 1.5V between the rising and falling edges.

A.C. Parameters for ICC Bus

V_{CC} = 5V ±5%; T_C = 0°C to +85°C

Symbol	Parameter	Ref. Fig.	Min (ns)	Max (ns)	Notes
tic	ICLK Period	35	60		(Note 1)
t40	ICLK High Time	35	20		
t41	ICLK Low Time	35	20		
t42	ICLK Rise Time	35		10	
t43	ICLK Fall TIme	35		10	
t44	MBI3–MB10 Setup Time	36	8		
t45	MBI3–MB10 Hold Time	36	5		
t46	MB03–MBO0 VALID Low Delay	36		50	(Note 2)
t47	MB03–MBO0 VALID High-Z Delay	36	5	15	(Note 3)
t48	MB03–MBO0 VALID Low-Z Delay	36	12	25	(Note 3)

All parameters are given in nanoseconds.
TTL level timing is measured at 1.5V for both "0" and "1" levels.

NOTES:
1. MBI3–0 and MBO3–0 timing is tested at 150 ns cycle time.
2. This parameter is specified for 50 pF load.
3. Not 100% tested. Guaranteed by design characterization.

12.0 REGISTER SUMMARY

82489DX registers can be located at any 1 Kbyte boundary in either memory or I/O space for as far as the 82489DX architecture itself is concerned. From a platform standard point of view, it is recommended to locate all 82489DX Local Units in memory space at address 0xFEE0—0000. It is further recommended that all 82489DX I/O Units also be located in memory space; I/O Unit 1 at address 0xFEC0—0000, I/O Unit 2 (if present) at address 0xFEC0—1000, and so on. Chip select for the 82489DX should be based on a full decode of address pins A31–A10.

All directly accessible 82489DX registers are 32 bits wide and are aligned at 128-bit boundaries. The register being accessed is determined by bits 4 through 9 of the address. This is listed in the tables below.

Addresses not listed are reserved by the architecture. The tables also show whether the register is readable and/or writable by software, and what the side effects are of software accessing the register.

After reset, all registers are initialized to all zeroes with the following exceptions, The Local Unit ID field is initialized with data present on the 8 LSB address pins. The Mask bit is initialized to 1 ("masked" state) in all entries in both the local vector table and the redirection table.

For the I/O Unit, only the I/O register select and I/O window registers are directly accessible in the address space. The other I/O unit registers are accessed indirectly through the select and window register.

4

I/O Unit Registers

Register	Address (9:4)	SW	Side Effects
I/O Register Select	00 0000	W	
I/O Window Register	00 0001		

Register	I/O Reg Select (7:0)	SW	Side Effects
I/O Unit ID Register	0000 0000	rw	
Version Register	0000 0001	r	
Redirection Table [0] (31:0)	0001 0000	rw	
Redirection Table [0] (63:32)	0001 0001	rw	
Redirection Table [1] [31:0]	0001 0010	rw	
Redirection Table [1] [63:32]	0001 0011	rw	
Redirection Table [2] (31:0)	0001 0100	rw	
Redirection Table [2] (63:32)	0001 0101	rw	
Redirection Table [3] [31:0]	0001 0110	rw	
Redirection Table [3] [63:32]	0001 0111	rw	
Redirection Table [4] (31:0)	0001 1000	rw	
Redirection Table [4] (63:32)	0001 1001	rw	
Redirection Table [5] [31:0]	0001 1010	rw	
Redirection Table [5] [63:32]	0001 1011	rw	
Redirection Table [6] (31:0)	0001 1100	rw	
Redirection Table [6] (63:32)	0001 1101	rw	
Redirection Table [7] [31:0]	0001 1110	rw	
Redirection Table [7] [63:32]	0001 1111	rw	
Redirection Table [8] (31:0)	0010 0000	rw	
Redirection Table [8] (63:32)	0010 0001	rw	
Redirection Table [9] [31:00]	0010 0010	rw	
Redirection Table [9] [63:32]	0010 0011	rw	
Redirection Table [10] (31:00)	0010 0100	rw	
Redirection Table [10] (63:32)	0010 0101	rw	
Redirection Table [11] [31:0]	0010 0110	rw	
Redirection Table [11] [63:32]	0010 0111	rw	
Redirection Table [12] (31:0)	0010 1000	rw	
Redirection Table [12] (63:32)	0010 1001	rw	
Redirection Table [13] [31:0]	0010 1010	rw	
Redirection Table [13] [63:32]	0010 1011	rw	
Redirection Table [14] (31:0)	0010 1100	rw	
Redirection Table [14] (63:32)	0010 1101	rw	
Redirection Table [15] [31:0]	0010 1110	rw	
Redirection Table [15] [63:32]	0010 1111	rw	

LOCAL UNIT REGISTERS

Registers	Address (9:4)	SW	Side Effects
Local Unit ID Register	00 0010	rw	
Version Register	00 0011	r	
Reserved	00 0100		
Reserved	00 0101		
Reserved	00 0110		
Reserved	00 0111		
Task Priority Register	00 1000	rw	mask intr dispense
Reserved	00 1001		
Reserved	00 1010		
EOI Register	00 1011	rw	prioritization cycle
Remote Register	00 1100	r	
Logical Destination Reg.	00 1101	rw	
Destination Format Reg.	00 1110	rw	
Spurious Vector Register	00 1111	rw	
ISR (31:0)	01 0000	r	
ISR (63:32)	01 0001	r	
ISR (95:64)	01 0010	r	
ISR (127:96)	01 0011	r	
ISR (159:128)	01 0100	r	
ISR (191:160)	01 0101	r	
ISR (223:192)	01 0110	r	
ISR (255:224)	01 0111	r	
TMR (31:0)	01 1000	r	
TMR (63:32)	01 1001	r	
TMR (95:64)	01 1010	r	
TMR (127:96)	01 1011	r	
TMR (159:128)	01 1100	r	
TMR (191:160)	01 1101	r	
TMR (223:192)	01 1110	r	
TMR (255:224)	01 1111	r	
IRR (31:0)	10 0000	r	
IRR (63:32)	10 0001	r	
IRR (95:64)	10 0010	r	
IRR (127:96)	10 0011	r	
IRR (159:128)	10 0100	r	
IRR (191:160)	10 0101	r	

Registers	Address (9:4)	SW	Side Effects
IRR (223:192)	10 0110	r	
IRR (255:224)	10 0111	r	
Intrpt Comnd Reg. (31:0)	11 0000	rw	send interrupt
Intrpt Comnd Reg. (63:32)	11 0001	rw	
Local Vector Table [timer]	11 0010	rw	
Reserved	11 0011		
Reserved	11 0100		
Local Vector Table [local int 0]	11 0101	rw	
Local Vector Table [local int 1]	11 0110	rw	
Reserved	11 0111		
Initial Count Register	11 1000	rw	
Current Count Register	11 1001	r	
Reserved	11 1010		
Reserved	11 1011		
Reserved	11 1100		
Reserved	11 1101		
Divider Configuration Reg.	11 1110	rw	
Reserved	11 1111		

NOTE:
Address space 101000 to 101111 and 111111 are reserved

13.0 TIMING DIAGRAMS

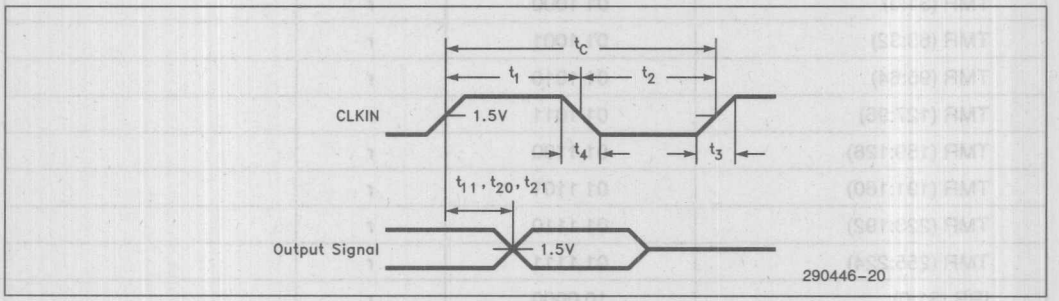


Figure 30. Output Waveform

13.0 TIMING DIAGRAMS (Continued)

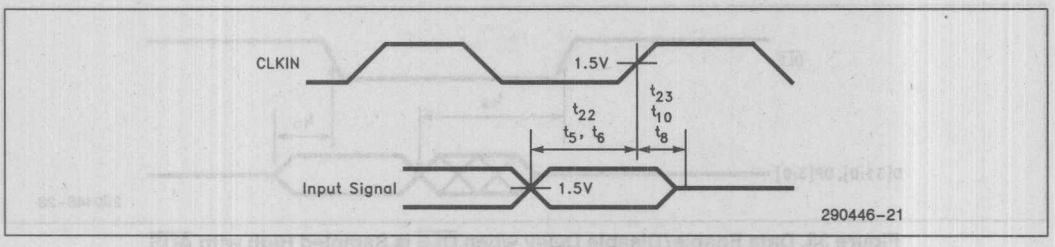


Figure 31. Input Waveforms

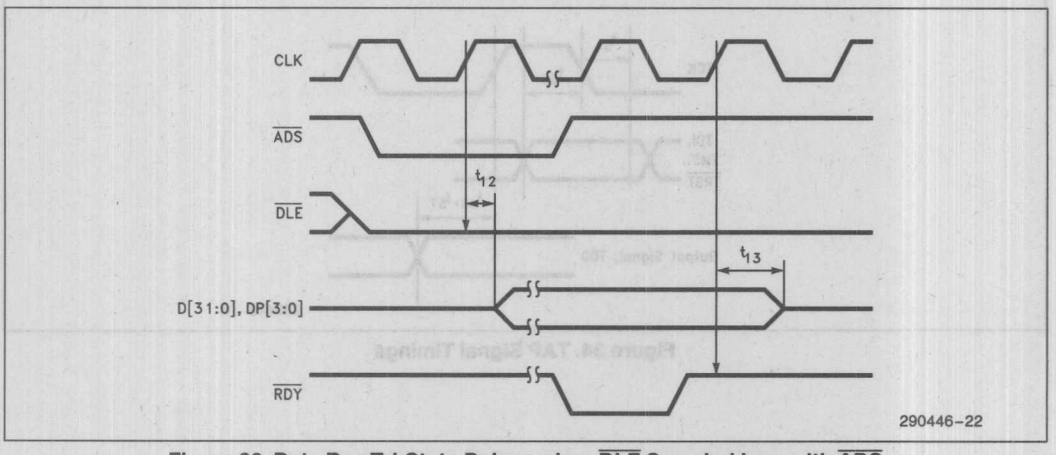


Figure 32. Data Bus Tri-State Delays when $\overline{DLE}$ Sampled Low with $\overline{ADS}$

4

13.0 TIMING DIAGRAMS (Continued)

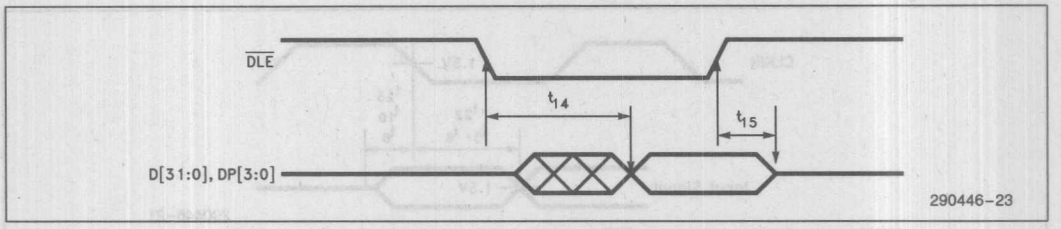


Figure 33. Data Enable/Disable Delay when $\overline{DLE}$ is Sampled High with $\overline{ADS}$

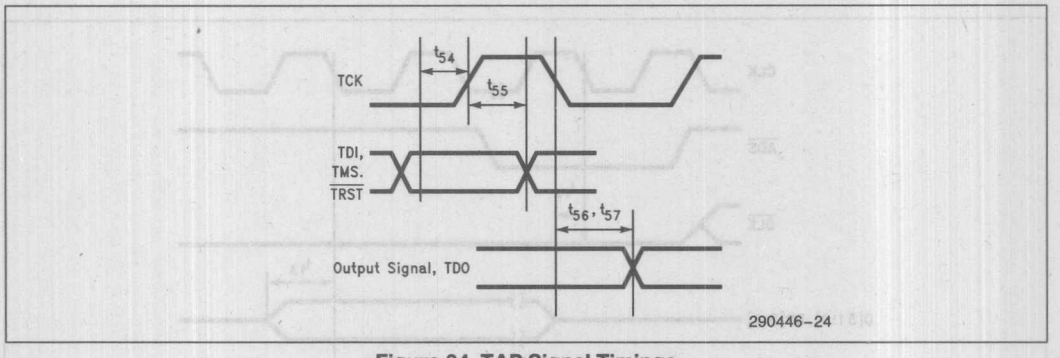


Figure 34. TAP Signal Timings

13.0 TIMING DIAGRAMS (Continued)

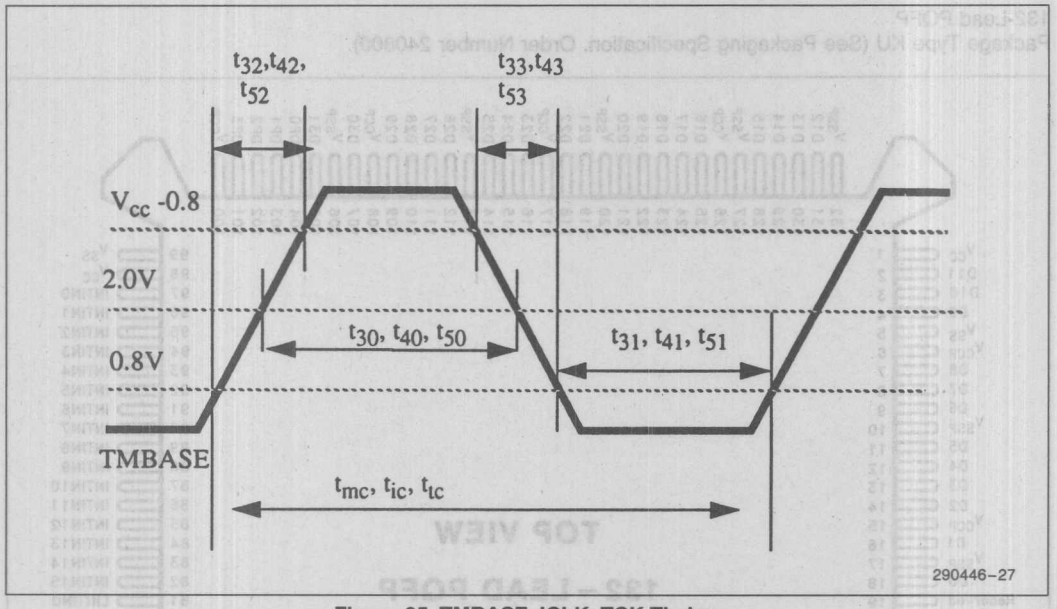


Figure 35. TMBASE, ICLK, TCK Timing

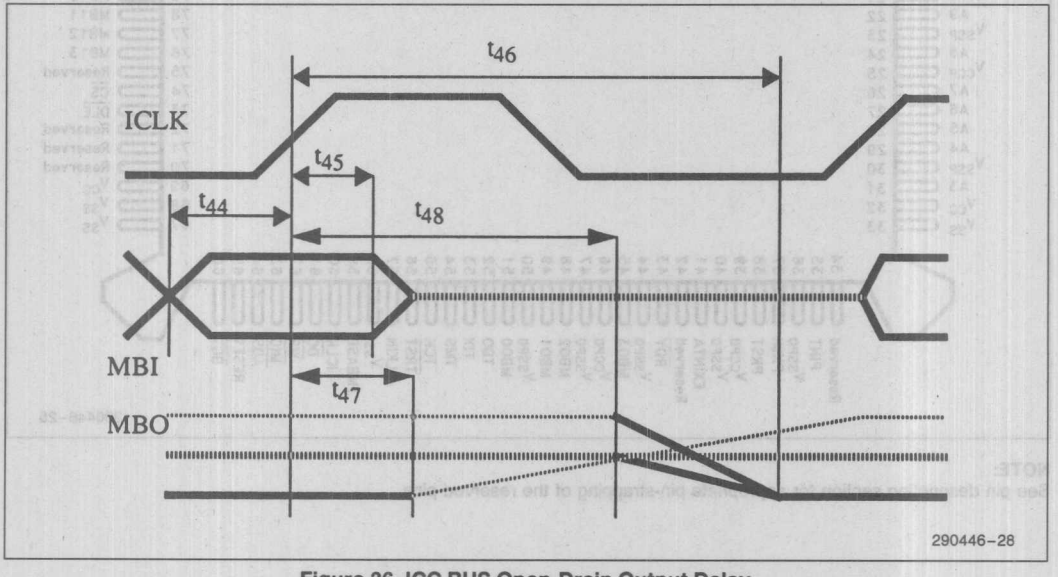
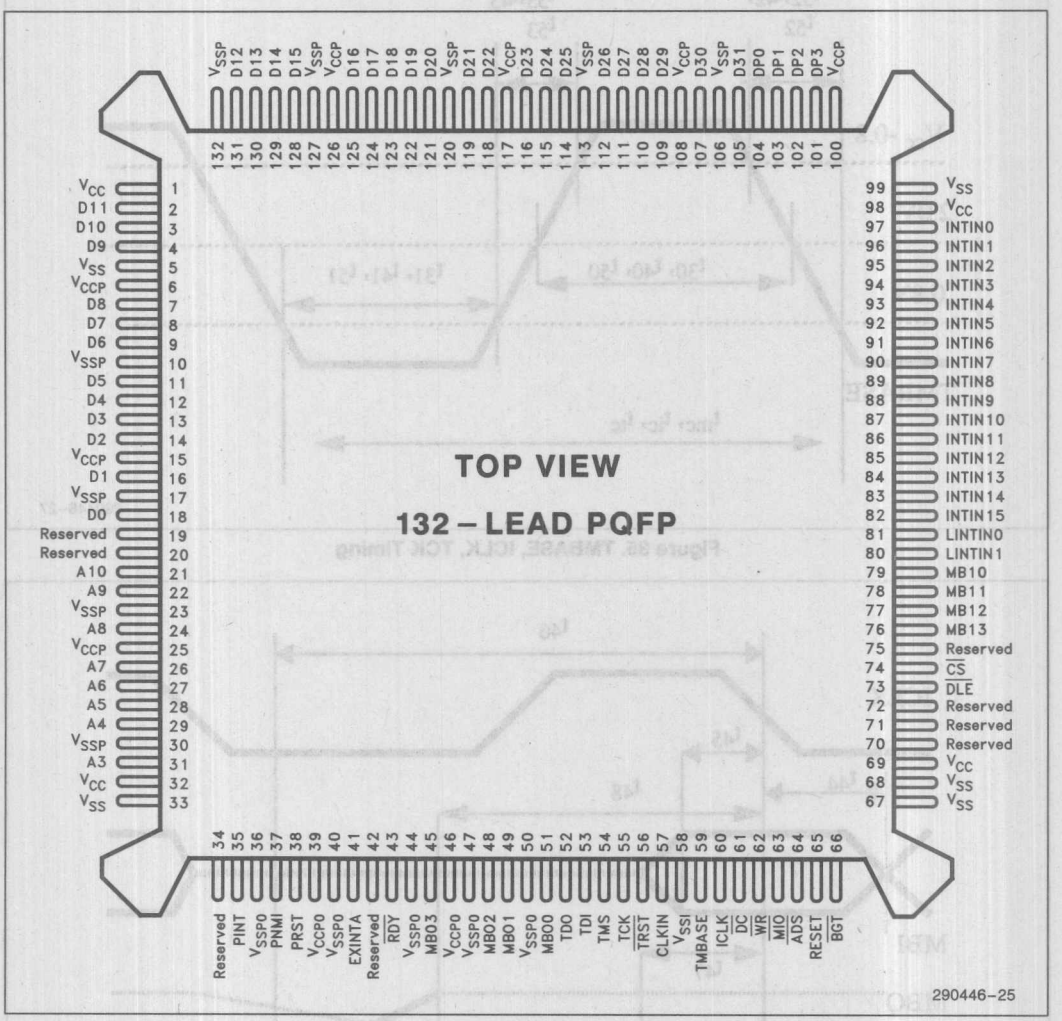


Figure 36. ICC BUS Open-Drain Output Delay

4

14.0 PACKAGE PIN-OUT

132-Lead PQFP
Package Type KU (See Packaging Specification. Order Number 240800)

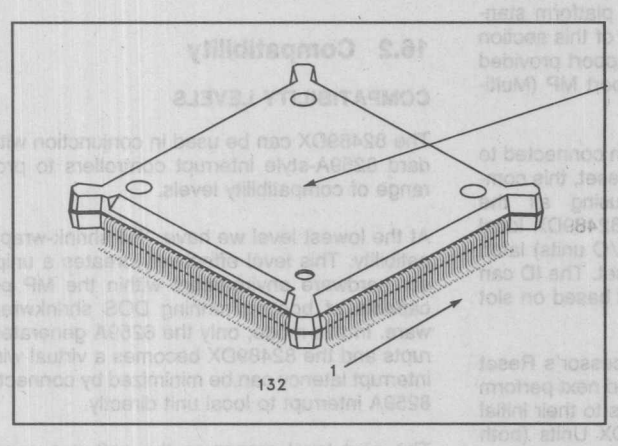


NOTE:
See pin description section for appropriate pin-strapping of the reserved pins.

15.0 PACKAGE THERMAL SPECIFICATION

The 82489DX is specified for operation when the case temperature is within the range of 0°C to +85°C. The case temperature may be measured in any environment, to determine whether the device is within the specified operating range.

The PQFP case temperature should be measured at the center of the top surface opposite the pins, as shown in Figure below.



290446-26

Plastic Quad Flat Pack (PQFP)

PQFP Package Thermal Characteristics

Parameter	Thermal Resistance—°C/W					
	Air Flow Rate (Ft./Min)					
	0	200	400	600	800	1000
θ Junction to Case	8	8	8	8	8	8
θ Junction to Ambient	32.5	25.5	20	18.5	16	15

NOTES:
1. Table above applies to 82489DX PQFP plugged into a socket or soldered directly into the board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.

4

16.0 GUIDELINES FOR 82489DX USERS

16.1 Initialization

This section outlines one possible initialization scenario. Other scenarios are certainly possible, and one would be selected as part of a platform standard initialization scheme. The intent of this section is to illustrate that the initialization support provided by the 82489DX is adequate to support MP (Multiprocessor) system initialization.

Each 82489DX has a RESET input pin connected to a common Reset line. Upon system reset, this common reset line is activated, causing all the 82489DXs to go through reset. All 82489DX local units (note: only local units and not I/O units) latch their ID from their address bus on reset. The ID can be provided by the bus control agent based on slot number.

The local units next assert their processor's Reset pin, holding the processor in reset, and next perform their internal reset, setting all registers to their initial state. The initial state of all 82489DX Units (both local and I/O units) is "all masks set" and all Local Units disabled; registers are otherwise initialized to zero. Note that the PINT and PNMI output pins are in tri-state mode when the local unit is disabled. After this, each 82489DX local unit will deassert its processor's Reset pin, allowing the processors to come out of reset and perform self test and start executing initialization code.

Note that while connecting PRST pin it should be noted that whenever PRST pin is activated by 82489DX either because of software reset message or hardware reset, the 82489DX itself is reset. It should be taken care in the cases of Warm reset where only processors need to be reset and not the interrupt controller. In brief, the usage of PRST depends upon the system requirement on various reset.

Somewhere in this code sequence, the processors that are "alive" will enable their 82489DX local units, and attempt to force all the other processors back into Reset. Forcing the other processors into reset is performed by sending them the inter-processor interrupt with Destination Mode = "Physical", Delivery Mode = "Reset", Trigger Mode = "Level", Level = "1", and Destination Shorthand = "All Excl Self". Only the first processor to get the ICC bus will succeed in sending this signal and reset all other 82489DXs and their processors. The other processors are kept in reset until such time that an MP operating system decides they can become active again. The only running processor next performs the rest of system initialization.

Eventually, an MP operating system will be booted at which time the operating system would send "deassert reset" interprocessor signals to activate the other processors in the system. A mechanism must be provided by the platform that allows the added processors to differentiate the very first reset from a subsequent one.

16.2 Compatibility

COMPATIBILITY LEVELS

The 82489DX can be used in conjunction with standard 8259A-style interrupt controllers to provide a range of compatibility levels.

At the lowest level we have "PC shrink-wrap" compatibility. This level effectively creates a uniprocessor hardware environment within the MP platform capable of booting/running DOS shrinkwrap software. In this mode, only the 8259A generates interrupts and the 82489DX becomes a virtual wire. The interrupt latency can be minimized by connecting the 8259A interrupt to local unit directly.

The next level preserves the software compatible view of an 8259A but it allows more than one processor to be active in the system. This results in an asymmetrical arrangement, with one processor fielding all 8259A interrupts but with added inter-processor interrupt capability. In this mode, 82489DX "merges" 8259A interrupts with inter-processor interrupts. Existing I/O drivers would be bound to the compatible CPU and interface directly with the 8259A.

At the next compatibility level, 8259A compatible drivers can be mixed with native 82489DX drivers. Devices can generate interrupts at either 8259A or an 82489DX. This provides for partial symmetry as individual drivers migrate from the 8259A to native 82489DXs.

Another 8259A compatible point can be defined for MP systems. Each processor could have its own compatible 8259A controllers, allowing multiple processors to run compatible I/O drivers, but statically spreading the load across the available processors.

82489DX/8259A INTERACTION

The principle of compatible operation is very straightforward; the 82489DX(s) become a virtual wire connecting the 8259A's INT output through to the processor, while at the same time making 8259A visible to the processor.

The two connection schemes described only differ in the number of 82489DX(s) (one or two) that are located in the path from the 8259A to the processor. In the one 82489DX example illustrated in Figure 37, the INT output of the 8259A connects to one of the Interrupt Input pins of the 82489DX through an edge generation logic. This could be an interrupt pin on the 82489DX's I/O unit or local unit; assume a local interrupt input is used. The Local Vector Table entry for the interrupt pin that connects to the 8259A is set up with a Delivery Mode of "ExtINT" and edge trigger mode. This indicates that the interrupt is generated by an external controller. The processor's INT pin connects to the 82489DX PINT pin.

This setup enables the 82489DX local unit to detect assertions (up-edges) of the 8259A's INT output pin and pass this on to the processor's INT input. 82489DX asserts ExtINTA pin along with (one clock prior to) PINT pin to indicate "8259" interrupt. When the processor performs its INTA cycle the 82489DX itself does not respond other than deasserting PINT to the processor. At the third clock after $\overline{ADS}$ in the second bus cycle of INTA cycle ExtINTA is deasserted. External logic should make use of the ExtINTA signal to make the INTA cycle visible to the 8259A and the 8259A should provide the vector. At the same time, the local unit considers the external request as delivered, and need not wait for the external 8259A's INT to be deasserted. *A new up-edge must be generated on the 8259A INT pin before the local unit will assert the processor's INT pin on behalf of the 8259A. External edge generation logic should be used for this.* Compatible software interacts directly with the 8259A.

The mechanism is essentially the same in the two-82489DX scheme. The difference is that the 8259A connects to an interrupt input pin of the 82489DX I/O unit in the I/O system. The Redirection Table entry for this pin is again programmed with an "ExtINT" Delivery Mode, and the (single) 82489DX destination local ID corresponding to the compatible DOS processor. Capturing the up-edges of the 8259A's INT pin by the 82489DX local unit now involves sending messages from the 82489DX I/O unit to the 82489DX local unit via the ICC bus. The "virtual wire" now includes messages over the ICC bus.

Adding inter-processor ICC interrupts (or any other 82489DX generated interrupts) to the compatible operation is accomplished by having the 82489DX internally OR the 8259A's INT request with any 82489DX interrupt request.

Before the 82489DX actually sends the interrupt signal to the processor, the 82489DX decides whether it does this for an 82489DX interrupt or whether it does this on behalf of the external controller. When the processor performs the corresponding INTA cycle, only the 82489DX knows whether it should respond with a vector, or whether the external 8259A should.

If the 82489DX needs to respond, then it will enable an externally implemented trap that prevents the 8259A from seeing the INTA cycle. If the 8259A needs to respond, then the 82489DX will not enable the INTA trap, and the INTA will be allowed to reach the 8259A. 82489DX implements this by asserting its EXTINTA pin to indicate external 8259A should respond with the vector. The 82489DX local unit controls the INTA trap via its "ExtINTA" output pin; the 82489DX does not actually provide the trap itself.

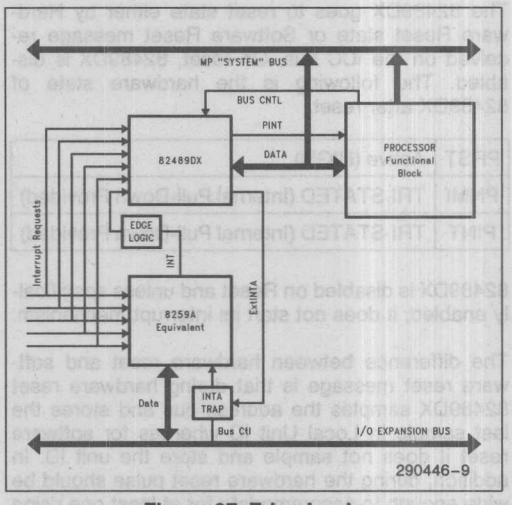


Figure 37. Edge Logic

82489DX/8259A DUAL MODE CONNECTION

In systems that can be booted either as a configuration with compatible 8259A or without, device interrupt lines are connected to both the Interrupt Request pins of the 8259A and Interrupt Input pins of the 82489DX with all interrupts either masked at the 82489DX or at the 8259A. Some EISA and Micro-Channel chip sets that include on-chip 8259As also have internally connected interrupt requests. For example, the 82357 (the ISP of the EISA chipset) generates timer and DMA chaining interrupts internally. These are not available as separate interrupts outside the ISP. In non compatible mode the ISP timers are not used, since each local 82489DX unit provides its own timer. Therefore, the ISP's 8259A is configured to mask out all interrupts except the DMA chaining interrupt which is configured in level-sensitive, auto EOI mode. This causes the 8259A's INT output to track the state of the internal DMA interrupt request. The 8259A's INT output is then connected to one of the 82489DX interrupt input pins programmed to generate a regular (i.e., not

"ExtINT") level-sensitive interrupt. The ISP 8259A then no longer functions as an external interrupt controller; it has been logically disabled, and it needs no interrupt acknowledge or EOI. The INTA and EOI cycles occur only at the 82489DX. It should be noted that 82489DX accepts only active high level/edge interrupt inputs. External programmable logic should take care of polarity reversal that may be needed in EISA system for sharing of interrupts.

16.3 Hardware Guidelines

82489DX HARDWARE STATE ON RESET

The 82489DX goes to reset state either by Hardware Reset state or Software Reset message received on the ICC bus. On reset, 82489DX is disabled. The following is the hardware state of 82489DX after reset.

PRST	Active (HIGH)
PNMI	TRI-STATED (Internal Pull-Down Provided)
PINT	TRI-STATED (Internal Pull-Down Provided)

82489DX is disabled on Reset and unless specifically enabled, it does not start its interrupt mechanism.

The difference between hardware reset and software reset message is that during hardware reset 82489DX samples the address bus and stores the last sample in Local Unit ID whereas for software reset it does not sample and store the unit ID. In addition, during the hardware reset pulse should be wide enough to accommodate for at least one rising and falling edge of ICLK. On hardware reset ExtINTA is held high.

PULL UP AND PULL DOWN RESISTORS

PNMI, PINT are tri-stated at power on and they are maintained in tri-state condition till the unit is enabled. Eventhough internal pull down resistor is provided on PNMI and PINT external additional pull down resistor may be needed depending upon the loading on these pins by external logic. The DC characteristics gives the control specification from which the value of resistor, if needed, can be calculated.

It should be kept in mind that the ICC bus being electrically open drain bus requires pull up resistors at the MBO pins. ICC bus output low current is just 4 mA.

PINT and ExtINTA Timings

It should be noted that for ExtINTA type of interrupts **PINT gets activated one clock after ExtINTA gets activated.** When getting deactivated, both PINT and ExtINTA gets deactivated at the same clock edge.

ExtINTA Timings

In the interrupt acknowledge cycle for External Interrupt control, 82489DX asserts ExtINTA. It decodes the type of cycle from CPU control signals like M/$\overline{IO}$, D/$\overline{C}$ and W/$\overline{R}$. CPU does two bus cycles back to back for interrupt acknowledge cycle. 82489DX maintains ExtINTA active throughout the first cycle. For next cycle (when the vector will be given by external 8259) after 82489DX senses the start of the cycle (by $\overline{ADS}$) 82489DX deactivates ExtINTA. External control logic may be inserting wait states to match the 8259 timings. Since 82489DX has no way of finding out the cycle completion, 82489DX deasserts ExtINTA before the second bus cycle gets completed. This should be kept in mind while using ExtINTA for external interrupt control logic.

82489DX AND MEMORY MAPPING

The 82489DX is a 32-bit high performance interrupt controller. It allows the CPU to do 32-bit read and write to it. By memory mapping 82489DX the system performance can be enhanced. It should be noted that 82489DX does not support pipelining. Eventhough 82489DX can be memory mapped, its functionality as an interrupt controller should be kept in mind while programming the virtual memory management control data structure. The caching policy for the page where 82489DX is mapped should also be done with the functionality of 82489DX in mind. For example, the reads to 82489DX should not be cached and writes should be write-through. Since 82489DX registers are aligned at 128-bit boundaries, memory mapping 82489DX with interleaved memory system should not be a problem.

JTAG CIRCUIT CONSIDERATIONS

The JTAG circuit is used for boundary scan test. The JTAG pins has a TCK, (JTAG clock), $\overline{TRST}$, (JTAG Reset), TDI, (Test Data Input), TMS, (Test Mode Select) and TDO, (Test Data Output).

The JTAG circuitry, if not used, should be properly deactivated so that it will not interfere in the normal functional operations. The JTAG can be inactivated in any one of the following ways:

1. JTAG inactivation through $\overline{TRST}$: The TMS, Test mode Select should be either left open (internal pull up is provided) or tied to V_{CC}. The $\overline{TRST}$ can be pulsed low (bring it low and after meeting the pulse width requirement bring it back high again) to keep the JTAG circuitry to idle state. The $\overline{TRST}$ pulse brings the JTAG circuitry to idle state and TMS being kept high maintains the JTAG circuitry in idle state.

2. JTAG inactivation through TCK: The TMS, Test mode select should be either left open (internal pull up is provided) or tied to V_{CC}. The TCK within 5 clocks brings the JTAG circuitry to idle state. The TMS, being held at logic high level, maintains the JTAG circuitry in idle state.

16.4 Programming Guidelines

The 82489DX register data structure contains different fields to specify the mode of operations and the options available within each mode. Since certain options are applicable to specific modes only (for example "Remote Read" mode applies only to interrupt command register, it does not have relevance to I/O unit's redirection tables) the following programming guidelines are provided.

UNIQUE ID REQUIREMENT

All the local units and I/O units hooked in a ICC bus should have unique ID before they can use the bus. This should be ensured by the programmer since for ICC bus arbitration the units (whether it is local unit or I/O unit) arbitrate with their unit ID.

For future compatibility, the Units should be assigned IDs starting with 0, 1, 2 etc. with the highest ID in the system being number of units minus 1. So in a four 82489DX system there are four local units and four I/O units. The ID starts with 0 and the highest ID in the system will be 7. Note that each unit should have different ID in the system.

ATOMIC WRITE READ TO TASK PRIORITY REGISTER

Normally, the task priority register is written with highest priority to mask certain low level interrupts before entering into critical section code. In a system where 82489DX is memory mapped the CPU may buffer this task priority register write to its on chip write buffer. The following scenerio can happen in such situation: CPU posts task priority register write to its on chip write buffer and enters into the critical code. A lower priority interrupt (which should not enter the critical code) interrupts the CPU before the write buffer gets flushed into task priority register). The CPU accepts the lower priority interrupt. To avoid the situation atomic write read to task priority register should be done. The read following write ensures that the write buffer is flushed to task priority register and the atomicity ensures that no interrupt will be accepted by the CPU during its write to task priority.

It should be noted that if the CPU does interrupt acknowledge cycle only after flushing the write buffers then the above situation may not arise.

CRITICAL REGIONS AND MUTUAL EXCLUSION

Each 82489DX has a single Interrupt Command Register that it uses to send interrupts to other processors. The programmer should make sure to synchronize access to this register. Specifically, 1). writing all fields of the register, 2). Sending the interrupt message (by writing the LSB register), and 3). waiting for Delivery State to become Idle again, should occur as a single atomic operation. For example, if interrupt handlers are allowed to send inter-processor interrupts, then interrupt dispensing to the processor must be disabled for the duration of these activities.

INTERRUPT COMMAND REGISTER PROGRAMMING SEQUENCE

The interrupt command register (31:0) has a side effect of sending interrupt once it is written. The destination is provided in the interrupt command register (63:32). So always interrupt command register (63:32) should be programmed before programming interrupt command register(31:0).

Program Interrupt Command Register (63:32)

↓

Program Interrupt Command Register (31:0)

INTERRUPT VECTOR

Two different interrupts should not be programmed with the same interrupt vector.

LOCAL AND I/O UNIT

Only Interrupt command register supports "Remote Read" Delivery mode. Local and I/O unit interrupts do not support "Remote Read".

ICR (INTERRUPT COMMAND REGISTER)

1. ExtINTA delivery mode is not supported for all destination shorthands.
2. "Remote Read" should always be programmed as "Edge" triggered interrupt.
3. "Remote Read" should always be programmed with physical destination mode (and not with Logical Destination mode). Broadcast addressing should not be used for Remote Read.
4. For "all incl Self" and "All exc. self" destination shorthands, "remote read" delivery mode should not be used.
5. For "all incl self" and "self" destination shorthands "Reset" delivery mode should not be used.

ICR (INTERRUPT COMMAND REGISTER) (Continued)

6. For "all exc self" destination shorthand if "Reset" delivery mode is used, it should be ensured at system level that only one processor executes this instruction at any time.
7. Messages could be sent out in "Logical" or "Physical" mode with destination ID of all 1's depending on the way Destination mode entry is programmed. In brief, "All incl self" and "All exc. Self" support both "Logical" and "Physical" addressing mode.
8. When destination shorthand (i.e., broadcast) is used with lowest priority destination mode, then even though all participates in arbitrating for destination, only the lowest priority gets the message. So even though the addressing is broadcast since the destination mode is lowest priority only one gets the message.
9. When destination shorthand (i.e., broadcast) is used with "Fixed" destination mode, then all the units get the message.

ISR/IRR/TMR

Bits 0–15 of IRR/ISR/TMR do not track interrupt. No interrupt of vector numbers from 0–15 can be posted. The total interrupt supported are 240. When reading the lowest 32 bits of these registers, 0 will be returned for the lower 16 bits.

FOCUS PROCESSOR

Focus processor is applicable only within the addressed units.

ExtINT Interrupt Posting

The external interrupt has no priority relationship with the 82489DX priority. But when posting an interrupt to the processor, if both an external interrupt and a 82489DX interrupt are pending, 82489DX could post either one to the processor. In 82489DX implementation, it would post external interrupt whenever there is no other 82489DX interrupt that can be posted to the processor. It should be also noted that External interrupts can not be masked by raising task priority. However, they can be masked by the mask bit in the table entry for the ("ExtINTA") external interrupt.

The extINT interrupts are specific in their characteristics in that they do not have any priority relationship with the rest of the interrupt structure. ISR and IRR bits in 82489DX are used to do the housekeeping functions for interrupt priority. Since extINT interrupts do not have any priority relationship, ISR and IRR bits are not maintained for external interrupts. As far as interrupt acceptance is concerned, if more than one extINT interrupts are directed towards a local unit, that local unit treats all the extINT interrupts directed to it as only one extINT interrupt. This leads to an important point that in a system not more than one interrupt should be programmed as extINT interrupt type with the same destination. It should be noted that there can be more than one extINT type of interrupt in a system with each having different local unit as destination.

Synchronizing Arb IDs

Initialization of an 82489DX's local unit ID is implementation dependent. In some platforms, power-on reset will latch the right values into the 82489DXs; in other platforms, unique IDs may be assigned by initialization firmware. In both cases the 82489DX I/O unit should be assigned unique ID by initialization firmware. The important point is that the 82489DXs are required to have unique IDs before they can use the bus, and in addition, all their Arb IDs must be "in sync". Synchronizing Arb IDs is accomplished as a side effect of a "deassert reset" interrupt command. This resets the (rotating) Arb ID to the (constant) unit ID; it assumes that all 82489DXs have their unique ID.

LOWEST PRIORITY

"Only once delivery" semantics for a group destination is guaranteed only if multiple fixed delivery of the same interrupt vector are not mixed. For lowest priority arbitration to work, all the arbitration ID of local 82489DXs in the system should be in sync. This means after local unit IDs are written in all local units (each ID should be different from other IDs) a RESET DEASSERT message should be sent in ALL INCLUSIVE mode. The RESET DEASSERT message should be sent before system is used for lowest priority arbitration. This ensures that all ARB IDs are also different. (Arb IDs are copied from local unit IDs during RESET DEASSERT message.)

The RESET DEASSERT message, if not sent, only one delivery semantics may not be guaranteed in the cases where lowest arbitration is used in the system.

DISABLING LOCAL UNIT

Once the 82489DX is enabled by setting bit 8 of spurious vector register to 1, the user should not disable the local unit by resetting the bit to 0. The result will put the local unit in an inconsistant state. The local unit can be disabled by sending "reset" interrupt message to the local unit.

ISSUING EOI

EOI, End of Interrupt issuing indicates end of service routine to 82489DX. The ISR bit which is set during INTA cycle gets cleared by EOI. This section discusses the relevence of EOI to the specific types of interrupts and its timing related to interrupt deassertion.

EXTERNAL INTERRUPTS AND EOI

External Interrupts should be programmed as edge type. INTA cycles to external interrupts are taken automatically as EOI by 82489DX. This is similar to AEOI, Automatic End of Interrupt of 8259A. So there is no need to issue EOI to 82489DX for external interrupt servicing. This is done to achieve software transparency in the compatible mode.

SPURIOUS INTERRUPTS AND EOI

Spurious interrupts do not have any priority relationship to other interrupts in the system. So IRR is not set for spurious interrupts. EOI should not be issued for spurious interrupts. It is advisable not to share the spurious interrupt with any vector. If spurious interrupt vector is shared with some other interrupt then while servicing issuing EOI depends on the source of interrupt. If the source is spurious interrupt (for which the corresponding IRR is not set) then EOI should not be used. If the source is a valid interrupt sharing the spurious interrupt vector (for which the IRR is set) then EOI should be issued.

NM AND EOI

For NM type of interrupt no IRR bit is set. So EOI should not be issued while servicing NMI type of interrupts.

TASK PRIORITY REGISTER

Task priority register is used to specify the priority of the task the processor is executing. In 8259 the priority is defined only among the interrupts that it handles. 82489DX goes farther ahead in handling priority. In multitasking system, in addition to device interrupts various tasks have different priority and 82489DX allows consideration of the priority at system level. The processor specifies the priority of the task it executes by writing to task priority register. Now any interrupts at and below the task priority will be masked temporarily till the task priority gets lowered. The masking granularity is at priority level. Out of 256 interrupt vectors 16 priority levels are specified and 16 vectors share one priority level. Since the masking granularity by the task priority register is at priority level, group of 16 vectors get masked when task priority register is increased by one level.

When task priority is at its minimum level of 0, interrupt vectors having level 1 to 16 are passed to CPU. Stated in other words, even when the task priority register is at its minimum (of level 0), interrupt vectors at level 0 will be masked. This means that the interrupt should not be programmed with vectors 0 to 15. So out of 256 interrupt vectors, only 240 interrupt vectors (vector 16 to 255) can be used in 82489DX.

ExtINT INTERRUPT AND TASK PRIORITY

ExtINT interrupt does not have any priority relationship with other interrupts or task priority register. So ExtINT interrupt can not be masked by raising task priority. They can be masked by writing to the vector table entry which corresponds to ExtINT interrupt.

REMOVING MASKS

When enabling units and removing Mask bits in situations where a device may already be injecting interrupts into the 82489DX system, the Mask in the Redirection Table should be removed last to ensure proper initial state (e.g., Remote IRR bit matching IRR in local unit).

DELIVERY MODE AND TRIGGER MODE

It is software's responsibility to make sure that Delivery Mode and Trigger Mode are set to meaningful combinations as listed below.

Delivery Mode	Trigger Mode
Fixed	edge/level
Lowest Priority	edge/level
Remote Read	edge
NMI	level
Reset	level
ExtINT	edge

Software is also responsible for not using meaningless Delivery Modes in Redirection Table entries and local Vector Table entries (e.g., use of Remote Read delivery mode).

4

ASSIGNING INTERRUPT VECTORS

Software has total control over the assignment of interrupt vectors to interrupt sources. The operating system writer should be aware of a number of things when doing this assignment.

Some processor architectures assign a predefined meaning to some of the vectors (i.e., entries in the interrupt table) as entry points to certain trap and exception handlers (e.g., divide error, invalid opcode, page fault, etc.). The programmer is strongly advised not to reuse these vectors.

The programmer must also be careful when using the same vector number to represent different interrupt sources (sharing vectors). This is especially true for level triggered interrupts. When multiple sources with different Redirection table entries share an interrupt vector, any of the sources deactivating its level signal will remove the interrupt request for all sources. Giving each interrupt source its interrupt vector in any case is the preferred approach.

SENDING INTER-PROCESSOR INTERRUPTS

Each 82489DX has a single Interrupt Command Register that it uses to send interrupts to other processors. It is software's responsibility to synchronize access to this register. Specifically, 1) writing all fields of the register, 2) sending the interrupt message (by writing the low register), and 3) waiting for Delivery State to become Idle again, should occur as a single atomic operation. For example, if interrupt handlers are allowed to send inter-processor interrupts, then interrupt dispensing to the processor must be disabled for the duration of these activities.

DELAY WITH LEVEL TRIGGERED INTERRUPTS

When a level triggered interrupt source deasserts its interrupt input, the destination will clear the interrupt's IRR bit only after receiving the message from the ICC bus. This introduces a small delay between the removal of the interrupt at the source and the removal of the interrupt at the processor. To avoid generating unnecessary interrupts, the interrupt handler should remove the interrupt at the source (at the device) as early as possible in the handler.

In any case, handlers should be able to deal with unnecessary interrupts.

RESET DEASSERT

A side effect of a reset deassertion message broadcast in the ICC bus is that all 82489DX local units reset their Arb ID to their unit ID. Interrupt commands that use the "self" Destination Shorthand do not generate a message on the ICC bus. If software only wants to generate the side effect of resetting Arb IDs, it should use a command with Logical Destination Mode and a Destination field containing all zeroes.

INTERRUPT MASKING

There are a number of levels at which interrupts can be masked, each resulting in a different behavior on interrupt delivery.

- First, interrupt injection (or deliver) can be masked by setting the Mask bit in the interrupt's Redirection Table or local Vector Table entry. These interrupts are ignored, no message is sent for them. Granularity is an individual interrupt.
- Second, each 82489DX can individually mask interrupt dispensing by raising its Task Priority to some level. This 82489DX will not dispense interrupts to its processor of this and lower priority unless it is currently the focus of the interrupt. Note again that the 82489DX is designed to operate as fully nested with non-specific EOI (to use existing 8259A terminology). There is no explicit interrupt mask (such as MR) and there is no notion of specific EOI.
- Third, each processor may provide a mechanism that masks all interrupt dispensing to it using the processor supplied instructions or status bits to do so. This does not interfere with lowest priority arbitration of the processor's 82489DX local unit.

CHANGING REDIRECTION TABLES

Redirection Tables are typically set up at initialization time. When modifying a Redirection Table entry "on the fly" the programmer must be aware of state kept at other 82489DXs relative to the interrupt being modified.

DEVICE DRIVERS WITH 82489DX

It is strongly recommended to read the device status registers before servicing the device. This is because if an edge triggered device deasserts its interrupt before interrupt acknowledge cycle (it should NOT) 82489DX will NOT give spurious vector. It will give genuine interrupt vector corresponding to the device. So, interrupt service routine should validate the interrupt request before servicing the device.

SYSTEM HARDWARE AND SOFTWARE DESIGN CONSIDERATIONS

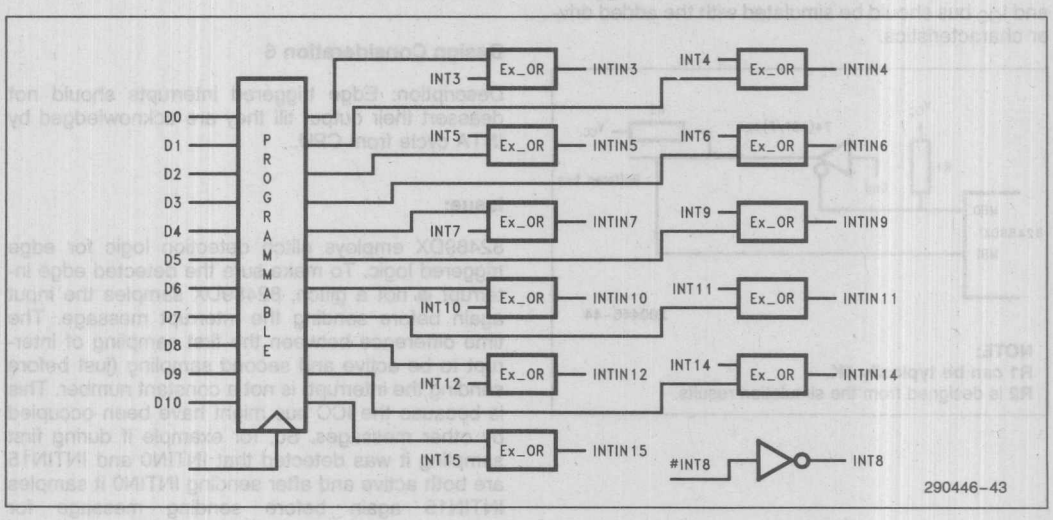
Design Consideration 1

Description: The following design consideration has to be taken care of when using ISP (82357) as external interrupt controller. 82489DX allows connecting external 8259 type interrupt controller at one of its inputs. The mode associated with the interrupt input which has 8259 connected to it is called ExtINTA mode. 82489DX allows only EDGE TRIGGERED programming option for ExtINTA mode. But in the case of 82357, the INT output from ISP stays high in case more than one interrupt is pending at its inputs. It does not always inactivate its INT output after INTA cycle. This will lead to a situation where ISP keeps the interrupt at high level continously and waits for INTA cycle. But since 82489DX expects an edge for interrupt sensing (for ExtINTA interrupts) it does not pass the interrupt to CPU and further interrupts are lost. So External circuitry should monitor the end of SECOND CYCLE of INTA cycle and force an inactive state at 82489DX's input. To avoid glitches at 82489DX input, this external logic should clear its output only at the end of second INTA cycle. It should be set by high going 82357 output. It should never be cleared by low going 82357 output. That is it should not follow 82357 output.

Design Consideration 2

Description: The following design consideration has to be taken care of when using 82489DX in EISA systems. EISA ISP(82357) chip integrates 8259A. It additionally allows sharing of interrupts. To facilitate this sharing it has a programmable register, ELCR (Edge / Level trigger control register) by which certain interrupt inputs can be programmed as edge (low to high except for RTC) or level (the level is active low). The determination of edge or level is done during initial configuration of EISA system by reading EISA add in boards from the interrupt description data structures. The solution is to have programmable logic at the interrupt inputs so that 82489DX is compatible with EISA ISP. This will introduce one more register and logic to support this. This should be an 11-bit programmable register and an array of ExOR logic (12 ExOR gates or equivalent PLD). The ISP allows programmability of the following interrupts.

INT3 INT4 INT5 INT6 INT7 INT9 INT10 INT11 INT12 INT14 INT15. In addition to the above 11 interrupts, it fixes **INT8** to be active low edge triggered interrupt. INT8 is the only case where it is active low edge triggered type. So the following logic can be used to add programmability in 82489DX based EISA system. Before connecting these 11 interrupt lines directly (#INT8 which is from Real Time Clock is always active low edge triggered. #INT8 can be passed through an inverter since there is no need for programmability) to the 82489DX they should pass through an array of 11 Ex__OR gates. One input of Ex__OR gate connects to the corresponding INT pin and other input connects to a bit of programmable register. The output of Ex__OR gate is connected to 82489DX. The idea of Ex__OR is to use as a controlled inverter.

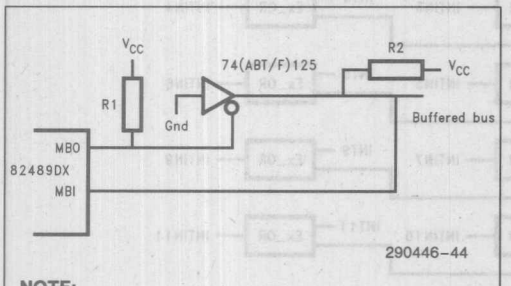


INTIN are the interrupt inputs to the 82489DX and INT are the system interrupt. The Ex_OR gating register is programmed after EISA configuration is found from add in boards as how these interrupt lines are going to be used in that particular configuration. If a particular input is edge triggered, then the corresponding bit in the register is written with 0. If a particular input is level triggered, then the corresponding bit in the register is written with 1.

8259 by itself does not have polarity control whereas 8259 when implemented in EISA chipsets have the polarity control. Similarly APIC does not have by itself polarity register. So polarity register should be programmed as a part of system BIOS and not APIC BIOS.

Design Consideration 3

I_{CC} bus drive is an open drain bus with drive capacity of 4 mA only. Since data is transmitted at each I_{CC} clock, the "charging" of I_{CC} bus should be fast enough to ensure proper logic level at each clock edge. The I_{CC} bus needs pull up resistors since it is open drain bus. Since the drive is only 4 mA, the pull up resistor value can not be less than 5V/4 mA. This being the limit of the resistor value, the length and the characteristics of the I_{CC} trace forces a capacitance value. Both the resistor and capacitance brings a RC time constant to the I_{CC} bus waveform. So, Electrical consideration has to be given to and practice of controlled impedence should be exercised for layout of the I_{CC} bus. The length of the trace should be kept as minimum as possible. If the length of the I_{CC} bus can't be kept less, than say 6 inch, because of mechanical design of the system, the external line drivers should be added to I_{CC} bus and I_{CC} bus should be simulated with the added driver characteristics.



NOTE:
R1 can be typically 1K.
R2 is designed from the simulation results.

Design Consideration 4

This is related to ADS#, BGT# and CS# timings. For bus cycles not intended for 82489DX, (CS# = 1 where 82489DX is supposed to sample it), any change in CS# line while the ADS# is still active, may erroneously cause a RDY# returned from 82489DX. Anomolous behavior may result if for BGT# ties low cases

a) BGT# goes away just one clock after ADS# or
b) ADS# is still active, and CS# changes during this period.

For other cases anomolous behavior results if CS# changes when ADS# is still active. The following considerations are important from timing point of view. Always limit the pulse width of 82489DX ADS# to one CLKIN. Also avoid changing levels on BGT#/CS# line, when ADS# is active for cases being identified as BGT# tied low (BGT# sampled low when ADS# goes active). Also avoid changing levels on CS# line when BGT# is active.

Design Consideration 5

82489DX does not recognize the interrupt when an edge occurs at the interrupt input pin while interrupt is masked. When later it is unmasked there is no further edge and so 82489DX never passes that forgotten edge and that interrupt channel becomes unusable after that.

The recommendation is that first 82489DX should be unmasked and then the device interrupt should be enabled in the device register. By this, software can ensure that always an edge will occur after an interrupt is unmasked.

Design Consideration 6

Description: Edge triggered interrupts should not deassert their output till they are acknowledged by INTA cycle from CPU.

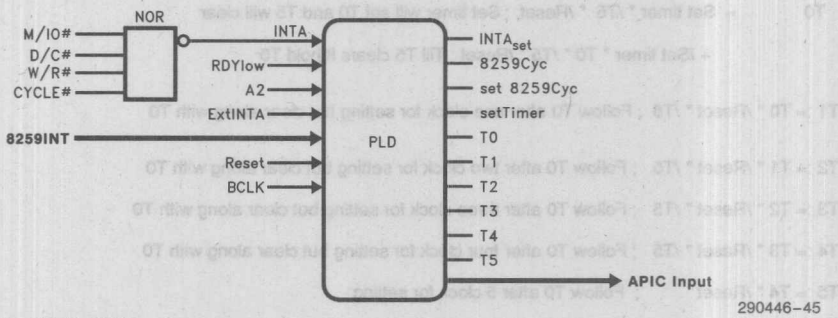
Issue:

82489DX employs glitch detection logic for edge triggered logic. To make sure the detected edge interrupt is not a glitch, 82489DX samples the input again before sending the interrupt message. The time difference between the first sampling of interrupt to be active and second sampling (just before sending the interrupt) is not a constant number. This is because the ICC bus might have been occupied by other messages. So, for example if during first sampling it was detected that INTIN0 and INTIN15 are both active and after sending INTIN0 it samples INTIN15 again before sending message for

INTIN15. But between this time ICC bus might have been occupied by other messages. So even if an edge triggered interrupt is held active high for a really long time and then brought low before INTA cycle, it is considered as a glitch. Because it may happen that the second sampling occurred just when the interrupt line got low.

Once the glitch detection circuitry found this "glitch", it goes back to the state where it will start sampling and waiting for an active edge to occur. This takes more than one clock cycle (CLK) and if the "glitch interrupt" generates an edge before that time after the second sampling of low level is done, then the edge is lost forever.

Since the time when the second sampling is done is unknown, the best way is to make sure the edge triggered interrupts do not deassert their outputs till they are acknowledged by INTA cycle from CPU. It is found that in some cases 8259 can generate brief active low pulses on its output. So the glue logic between 8259 and 82489DX input pin should make sure that 82489DX input pin is clear only after getting second interrupt acknowledge cycle. The glue logic should not just follow the 8259 output. Put in other words, after interrupt acknowledge cycle to 8259, if the 8259 input is seen active high, it should generate an edge at 82489DX input. Moreover, even if 8259 output goes low the glue logic should not lower its output since the only time when the glue logic can deassert its output is when it finds an interrupt acknowledge cycle for 8259. The following PLD equations and schematics serves as an example for the glue logic between 8259 and 82489DX.



BCLK is the standard EISA BCLK.
CYCLE# is an external signal to indicate ADS# is asserted earlier and RDY low is not yet returned.

4

```
APIC input =    /T0  * /Reset * /APIC input * 8259INT  * INTA set  ; Sample 8259 interrupt
             + /T0 * /Reset * APICinput  * INTA set          ; Hold till it is cleared by delayed interrupt acknowledge
             + / INTA set * 8259INT

Set 8259Cyc  = /Reset * INTA * ExtINTA    ; This INTA cycle is for 8259

8259Cyc      = set8259Cyc * /T0 * /Reset   ; Set 8259cyc will set 8259 cycle and T0 will clear it
             + /T0 * /Set8259Cyc * 8259Cyc * /Reset ;  Hold 8259 cycle till T0 clears it

INTA set     = /Reset * /INTA set * INTA ; wait for very first INTA cycle after reset
             + /Reset * INTA set ; once first INTA cycle after Reset is found, set the INTA set

Set timer    = 8259cyc * /A2 * /RDYlow    ; Start the timer at end of second INTA cycle

T0           = Set timer * /T5  * /Reset  ; Set timer will set T0 and T5 will clear
               + /Set timer * T0 * /T5 * /Reset ; Till T5 clears it hold T0

T1 := T0 * /Reset * /T5  ; Follow T0 after one clock for setting but clear along with T0

T2 := T1 * /Reset * /T5  ; Follow T0 after two clock for setting but clear along with T0

T3 := T2 * /Reset * /T5  ; Follow T0 after three clock for setting but clear along with T0

T4 := T3 * /Reset * /T5  ; Follow T0 after four clock for setting but clear along with T0

T5 := T4 * /Reset        ; Follow T0 after 5 clock for setting
```

290446-46

NOTES:
T1, T2, T3, T4 and T5 are clocked signals and others are combinatorials.
This circuit and PLD equations are given for concept clarification purpose. They are not tested.
$INTA_{set}$ is needed so that some 8259 logic at power on activates its INT output to 1 and it deactivates its output after only 8259 initialization (which should happen after APIC initialization) and since APIC needs to detect rising edge at 8259, it is essential to follow the 8259 until first interrupt. This is the only occasion 8259 output will be just followed.

DIRECTIONS FOR EASY MIGRATION TO FUTURE INTEGRATED APIC

The following are the software programming directions Intel strongly recommends for easy migration from 82489DX to integrated APIC. The audience to this portion of the document are both hardware designers and firmware developers for APIC based systems. In the following discussions, the APIC BIOS is viewed functionally as two subsections 1) APIC BIOS which are all interrupt vector, priority, interrupt distribution related functions and the remaining portion of BIOS which is referred to part of system BIOS which is responsible for interrupt polarity programming, starting next processor, etc.

Note that the names APIC BIOS and APIC DRIVER are interchangeably used in the following discussion. Different Operating systems refer such functional module differently.

Consideration 1

Question: The logical destination register in future implemented APIC may have only 8 MSBs defined and 82489DX has 32 bits specified. Will this hinder binary level compatibility?

Response: In logical destination (flat addressing mode) 82489DX can go up to 32 CPUs whereas future APIC can go up to 8 CPUs with flat logical addressing mode. ***For binary compatibility, it is strongly recommended that 82489DX software use ONLY 8 MSB of logical destination register.***

Consideration 2

Question: The present day MP systems with external control ports for starting next processor may program those external control ports for starting next processor. APIC DRIVER may use external control ports for starting next processor. In future implementations of APIC, the starting of next processors may use more refined mechanisms which may not use external control ports. Will this introduce compatibility problem?

Response: Again, the starting of next processor is really part of MP system DRIVER and depending of the mechanism used to start next processor it will vary. In future implementation if starting next processor is done using new mechanisms, the starting next processor portion of MP DRIVER will be changed accordingly. Even though this will not result in any change in the APIC DRIVER which deals with interrupt priority, distribution, etc., the corresponding change will be needed in the starting application processors portion of DRIVER.

One possible method of implementing software is using version register. Version register is different in 489DX and future implementations of APIC. Taking care of these differences, such as mechanism for starting next processor, should be possible using version register.

Consideration 3

Question: APIC architecture, by its nature, seems to misinterpret spurious interrupts as genuine interrupts. That is, if an edge triggered interrupt goes inactive before interrupt acknowledge cycle, APIC, instead of giving spurious interrupt vector, gives genuine interrupt vector. Is it true that this is not the case with 8259? If that is the case, drivers which do not check device status registers for servicing the device may work with 8259 but may not work with APIC. Is this a compatibility problem?

Response: No, this is not true. Even with 8259 there is a time window in which a similar thing can happen. For example if interrupt goes inactive just after first INTA cycle but before second INTA cycle 8259 will also signal this spurious interrupt as genuine interrupt. So drivers which do not check device status registers may also fail with 8259.

Our strong recommendation to device drivers is to read device status register before servicing the device. If the device status register indicates that there is no valid source of the interrupt, the service routine should just issue EOI and return. It should not service the device. This should take care of the new drivers that will be written for APIC. To coexist with 8259, the APIC interrupt input connected to 8259 will be programmed for virtual wire mode. In virtual wire mode, the time window of 8259 will apply. So the driver will behave same way as it was behaving with 8259.

Consideration 4

Question: EISA system has active low level polarity. 82489DX itself does not have polarity control register to support this EISA feature. Implementations using external polarity register may implement the polarity register at different address. Will this introduce a problem for achieving the goal of single binary?

Response: 8259 by itself does not have polarity control whereas 8259 when implemented in EISA chipset has the polarity control. Similarly APIC does not have by itself polarity register. When implemented in ESC chipset, it will have polarity control register. So polarity register should be programmed as a part of EISA BIOS and not APIC BIOS. Since system BIOS or EISA BIOS should be able to take charge of changes, if any, to polarity control register. APIC BIOS should not be affected by differences in the address for polarity register.

Consideration 5

Question: 8259 recognizes the interrupt when an edge occurs at the interrupt input pin even if the interrupt was masked. So when the interrupt input is later unmasked, the interrupt is posted to the CPU. 82489DX does not register this edge and if interrupt happens when the interrupt is masked 82489DX just ignores the interrupt. When later it is unmasked there is no further edge and so 82489DX never passes that forgotten edge and that interrupt channel becomes unusable after that.

Response: When the interrupt is masked, logically interrupt controller should ignore whatever happens there. It is strongly recommended that first 82489DX should be unmasked and then the device interrupt should be enabled. By this sequence, software can ensure that always an edge will occur at the APIC input only after the interrupt is unmasked.

Please contact Intel for platform level specification in Multiprocessor system design using APIC.

UPI-41AH/42AH
UNIVERSAL PERIPHERAL INTERFACE
8-BIT SLAVE MICROCONTROLLER

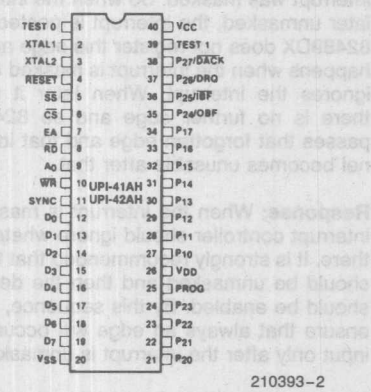
- **UPI-41: 6 MHz; UPI-42: 12.5 MHz**
- **Pin, Software and Architecturally Compatible with all UPI-41 and UPI-42 Products**
- **8-Bit CPU plus ROM/OTP EPROM, RAM, I/O, Timer/Counter and Clock in a Single Package**
- **2048 x 8 ROM/OTP, 256 x 8 RAM on UPI-42, 1024 x 8 ROM/OTP, 128 x 8 RAM on UPI-41, 8-Bit Timer/Counter, 18 Programmable I/O Pins**
- **One 8-Bit Status and Two Data Registers for Asynchronous Slave-to-Master Interface**
- **DMA, Interrupt, or Polled Operation Supported**
- **Fully Compatible with all Intel and Most Other Microprocessor Families**
- **Interchangeable ROM and OTP EPROM Versions**
- **Expandable I/O**
- **Sync Mode Available**
- **Over 90 Instructions: 70% Single Byte**
- **Available in EXPRESS — Standard Temperature Range**
- **int$_e$ligent Programming Algorithm — Fast OTP Programming**
- **Available in 40-Lead Plastic and 44-Lead Plastic Leaded Chip Carrier Packages**

(See Packaging Spec., Order #240800-001)
Package Type P and N

The Intel UPI-41AH and UPI-42AH are general-purpose Universal Peripheral Interfaces that allow the designer to develop customized solutions for peripheral device control.

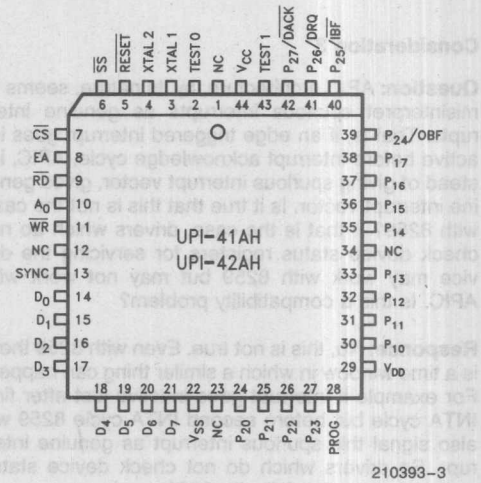
They are essentially "slave" microcontrollers, or microcontrollers with a slave interface included on the chip. Interface registers are included to enable the UPI device to function as a slave peripheral controller in the MCS Modules and iAPX family, as well as other 8-, 16-, and 32-bit systems.

To allow full user flexibility, the program memory is available in ROM and One-Time Programmable EPROM (OTP). All UPI-41AH and UPI-42AH devices are fully pin compatible for easy transition from prototype to production level designs.



210393-2

Figure 1. DIP Pin Configuration



210393-3

Figure 2. PLCC Pin Configuration

Refer to the 1994 Peripheral Components Handbook for the complete data sheet on this device.

October 1991
Order Number: 210393-007

UPI-C42/UPI-L42
UNIVERSAL PERIPHERAL INTERFACE
CHMOS 8-BIT SLAVE MICROCONTROLLER

- **Pin, Software and Architecturally Compatible with all UPI-41 and UPI-42 Products**
- **Low Voltage Operation with the UPI-L42**
  — Full 3.3V Support
- **Integrated Auto A20 Gate Support**
- **Two New Power Down Modes**
  — STANDBY
  — SUSPEND
- **Security Bit Code Protection Support**
- **8-Bit CPU plus ROM/OTP EPROM, RAM, I/O, Timer/Counter and Clock in a Single Package**
- **4096 x 8 ROM/OTP, 256 x 8 RAM 8-Bit Timer/Counter, 18 Programmable I/O Pins**
- **DMA, Interrupt, or Polled Operation Supported**
- **One 8-Bit Status and Two Data Registers for Asynchronous Slave-to-Master Interface**
- **Fully Compatible with all Intel and Most Other Microprocessor Families**
- **Interchangeable ROM and OTP EPROM Versions**
- **Expandable I/O**
- **Sync Mode Available**
- **Over 90 Instructions: 70% Single Byte**
- **Quick Pulse Programming Algorithm**
  — Fast OTP Programming
- **Available in 40-Lead Plastic, 44-Lead Plastic Leaded Chip Carrier, and 44-Lead Quad Flat Pack Packages**
  (See Packaging Spec., Order #240800, Package Type P, N, and S)

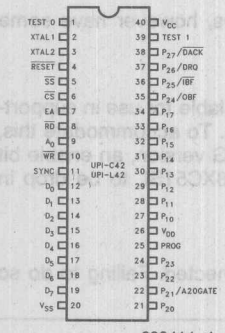
4

The UPI-C42 is an enhanced CHMOS version of the industry standard Intel UPI-42 family. It is fabricated on Intel's CHMOS III-E process. The UPI-C42 is pin, software, and architecturally compatible with the NMOS UPI family. The UPI-C42 has all of the same features of the NMOS family plus a larger user programmable memory array (4K), integrated auto A20 gate support, and lower power consumption inherent to a CHMOS product.

The UPI-L42 offers the same functionality and socket compatibility as the UPI-C42 as well as providing low voltage 3.3V operation.

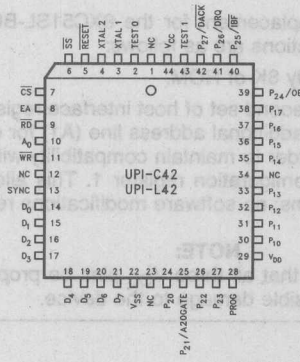
The UPI-C42 is essentially a "slave" microcontroller, or a microcontroller with a slave interface included on the chip. Interface registers are included to enable the UPI device to function as a slave peripheral controller in the MCS Modules and iAPX family, as well as other 8-, 16-, and 32-bit systems.

To allow full user flexibility, the program memory is available in ROM and One-Time Programmable EPROM (OTP).



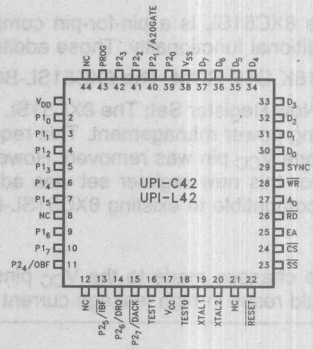
290414-1

Figure 1. DIP Pin Configuration



290414-2

Figure 2. PLCC Pin Configuration



290414-3

Figure 3. QFP Pin Configuration

Refer to the 1994 Peripheral Components Handbook for the complete data sheet on this device.

October 1993
Order Number: 290414-003

ADVANCE INFORMATION

8XC51SL/LOW VOLTAGE 8XC51SL KEYBOARD CONTROLLER

80C51SL — CPU with RAM and I/O; V_{CC} = 5V ±10%

81C51SL — 16K ROM Preprogrammed with SystemSoft Keyboard Controller and Scanner Firmware. V_{CC} = 5V ±10%.

83C51SL — 16K Factory Programmed ROM. V_{CC} = 5V ±10%.

87C51SL — 16K OTP ROM. V_{CC} = 5V ±10%.

Low Voltage 80C51SL—CPU with RAM and I/O; V_{CC} = 3.3V ±0.3V

Low Voltage 81C51SL— 16K ROM Preprogrammed with SystemSoft Keyboard Controller and Scanner Firmware. V_{CC} = 3.3V ±0.3V.

Low Voltage 83C51SL— 16K Factory Programmed ROM. V_{CC} = 3.3V ±0.3V.

Low Voltage 87C51SL— 16K OTP ROM. V_{CC} = 3.3V ±0.3V.

- **Proliferation of 8051 Architecture**
- **Complete 8042 Keyboard Control Functionality**
- **8042 Style Host Interface**
- **Optional Hardware Speedup of GATEA20 and RCL**
- **Local 16 x 8 Keyboard Switch Matrix Support**
- **Two Industry Standard Serial Keyboard Interfaces; Supported via Four High Drive Outputs**
- **5 LED Drivers**
- **Low Power CHMOS Technology**
- **4-Channel, 8-Bit A/D**
- **Interface for up to 32 Kbytes of External Memory**
- **Slew Rate Controlled I/O Buffers Used to Minimize Noise**
- **256 Bytes Data RAM**
- **Three Multifunction I/O Ports**
- **10 Interrupt Sources with 6 User-Definable External Interrupts**
- **2 MHz–16 MHz Clock Frequency**
- **100-Pin PQFP (8XC51SL)**
  100-Pin SQFP (Low Voltage 8XC51SL)

The 8XC51SL, based on Intel's industry-standard MCS® 51 microcontroller family, is designed for keyboard control in laptop and notebook PCs. The highly integrated keyboard controller incorporates an 8042-style UPI host interface with expanded memory, keyboard scan, and power management. The 8XC51SL supports both serial and scanned keyboard interfaces and is available in pre-programmed versions to reduce time to market. The Low Voltage 8XC51SL is the 3.3V version optimized for even further power savings. Throughout the remainder of this document, both devices will generally be referred to as 51SL.

The 8XC51SL is a pin-for-pin compatible replacement for the 8XC51SL-BG. It does, however have some additional functionality. Those additional functions are as follows:

1. 16K OTP ROM: The 8XC51SL-BG had only 8K of ROM.
2. New Register Set: The 8XC51SL adds a second set of host interface registers available for use in supporting power management. This required an additional address line (A1) for decoding. To accommodate this, one V_{CC} pin was removed. However, in order to maintain compatibility with the -BG version, an enable bit for this new register set was added in configuration register 1. This allows the 8XC51SL to be drop in compatible to existing 8XC51SL-BG designs; no software modifications required.

NOTE:

The changes made to the V_{CC} pins require that all three V_{CC} pins be properly connected. Failing to do so could result in high leakage current and possible damage to the device.

For the complete data sheet on this device, please call Intel's Literature Distribution Center (800) 548-4725.

October 1993
Order Number: 272271-002

iPLD610
FAST 16-MACROCELL CMOS PLD

Function, Pin, and JEDEC Compatible with EP600, EP610, EP610A, EP630, PALCE610, 85C060 and 5C060 PLDs

- t_{PD} 10 ns, 100 MHz Counter Frequency (w/Internal Feedback)
- I_{CC} = 105 mA max. @ 1 MHz
- Programmable Low-Power Option for "Standby" Operation; 20 μA Typ. in Standby Mode
- Clocking Speed Same as −7 ns PAL* (74 MHz w/External Feedback)
- 16 Macrocells with Programmable I/O Architecture (Register/Combinatorial). Registers Configurable as D/T/JK/RS Types
- Up to 20 Inputs (4 Dedicated and 16 I/O)
- 8 P-Terms, Selectable SOP Invert, Clear and OE P-Terms for Each Macrocell
- Programmable Clock System with 2 Synchronous Clocks and Asynchronous Clocking Option on all Registers
- Extensive Software and Programming Support via Intel and Third Party Tools
- 1-Micron CMOS* III-E EPROM Technology
- Programmable "Security Bit" Allows Total Protection of Proprietary Designs
- 100% Generically Tested Logic Array
- Available in 300-mil 24-Pin PDIP, CerDIP and 28-Pin PLCC Packages

(See Packaging Specifications Order Number #240800-001, Package Type N and P)

4

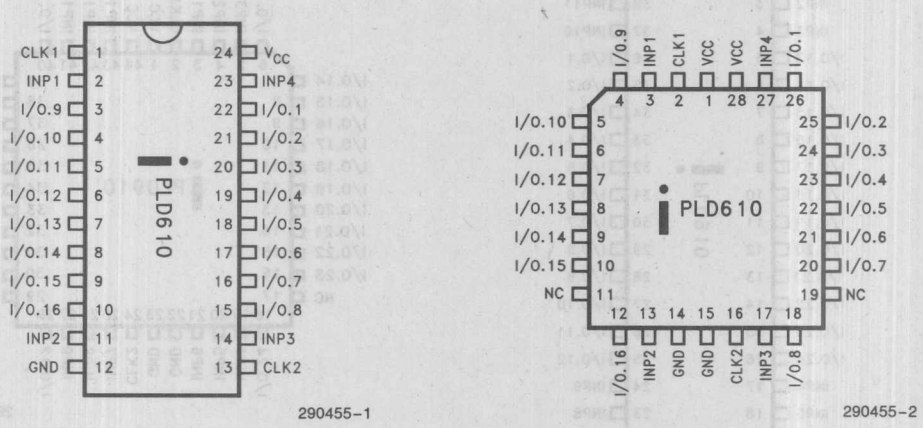


Figure 1. iPLD610 Pin Configurations

*PAL is a registered trademark of Advanced Micro Devices, Inc.

Refer to the 1994 Programmable Logic Handbook for the complete data sheet on this device.

October 1993
Order Number: 290455-002

iPLD910
FAST 24-MACROCELL CMOS PLD

Function, Pin, and JEDEC Compatible with EP900, EP910, EP910A, 85C090 and 5C090

- t_{PD} 12 ns, 62.5 MHz w/Feedback, Clock to Output 8 ns
- I_{CC} = 150 mA Max @ 1 MHz
- Programmable Low-Power Option for "Standby" Operation; 60 μA Typ. in Standby Mode
- 24 Macrocells with Programmable I/O Architecture (Register/Combinatorial). Registers Configurable as D/T/JK/RS Types
- Up to 36 Inputs (12 Dedicated and 24 I/O)
- 8 P-terms, Selectable SOP Invert, Clear and OE P-terms for Each Macrocell
- Programmable Clock System with 2 Synchronous Clocks and Asynchronous Clocking Option on all Macrocells
- Extensive Software and Programming Support via Intel and Third-Party Tools
- 1-Micron CMOS IIIE* EPROM Technology
- Programmable Security Bit Allows Total Protection of Proprietary Designs
- 100% Generically Tested Logic Array
- Available in 40-Pin PDIP, CERDIP and 44-Pin PLCC Packages

(See Packaging Spec., Order Number 240800, Package Type N and P).

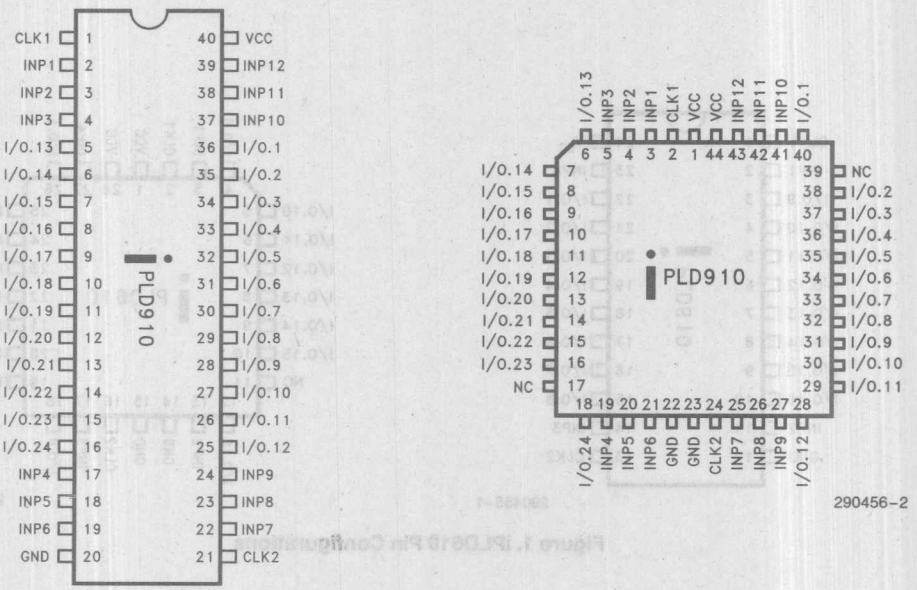


Figure 1. iPLD910 Pinout Diagrams

*CMOS is a patented process of Intel Corporation.

Refer to the 1994 Programmable Logic Handbook for the complete data sheet on this device.

September 1993
Order Number: 290456-002

iPLD22V10-10, -15, -25
HIGH PERFORMANCE 10-MACROCELL CMOS PLD

High-Speed Upgrade to Bipolar 22V10 and CMOS Equivalents

- t_{PD} 10 ns, 95.2 MHz with Feedback, 100 MHz with No Feedback
- Typical I_{CC} = 90 mA @ 15 MHz
- 12 Dedicated Inputs and 10 I/O Pins
- 10 Macrocells with Programmable I/O Architecture (Register/Combinatorial)
- Variable P-terms—Up to 16 per Macrocell, Selectable Output Polarity, Separate Output Enable P-term
- Global Asynchronous Clear and Synchronous Preset P-terms
- 1-Micron CHMOS IIIE EPROM Technology
- Programmable "Security Bit" Allows Total Protection of Proprietary Designs
- 100% Generically Tested Logic Array
- Available in 300-mil 24-Pin PDIP and 28-Pin PLCC Packages

  (See Packaging Spec., Order Number 240800, Package Type N and P)

4

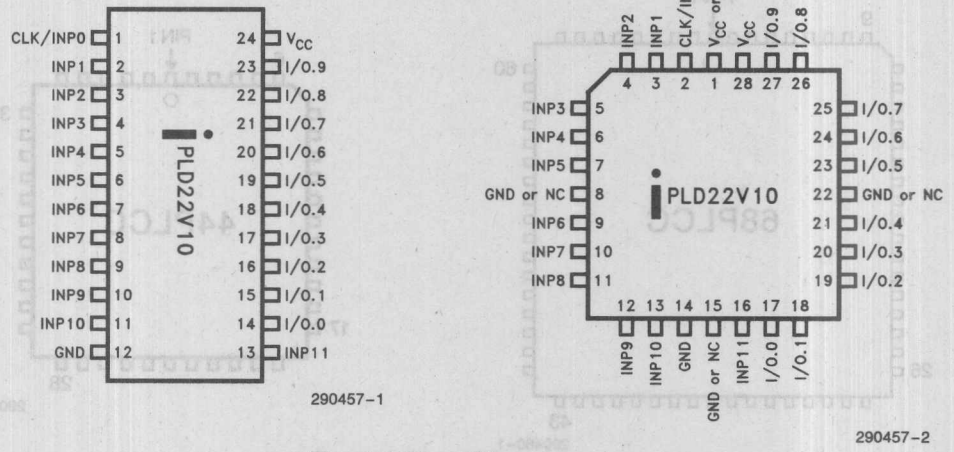


Figure 1. Pinout Diagrams

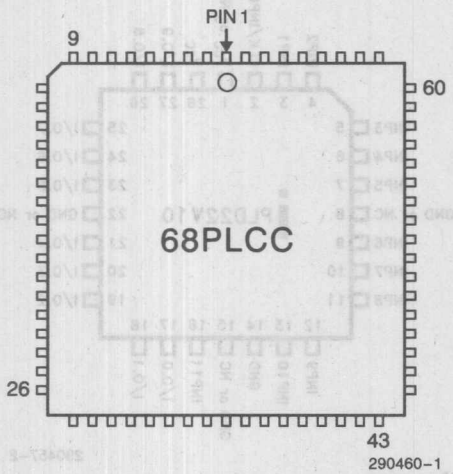
Log/IC is a trademark of ISDATA, Inc.

Refer to the 1994 Programmable Logic Handbook for the complete data sheet on this device.

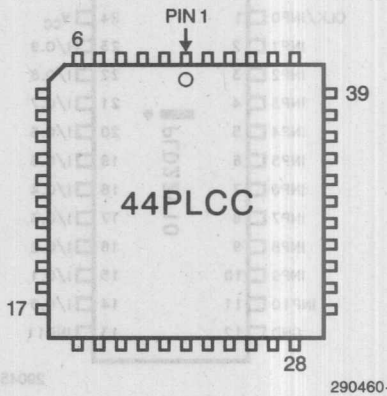
October 1993
Order Number: 290457-003

10 ns FLEXlogic FPGA Family WITH SRAM OPTION

- **High Performance FPGA (Field Programmable Gate Array)**
  - — Deterministic 10 ns Pin-to-Pin Propagation Delays
  - — 80 MHz System Clock Frequency
- **2,500 Equivalent Logic Gates or up to 5,120 Bits of SRAM**
- **0.8μ CMOS Technology**
  - — Power Management Options Minimize Active Power Consumption (1 mA/MHz)
  - — 1 mA Standby Version Available
- **JTAG 1149.1 Compatible Test Port**
  - — Supports Boundary Scan and In-Circuit Reconfiguration/ Programming
- **Four Configurable Function Blocks (CFBs) Linked by a 100% Connectable Matrix**
  - — Improves Fitting of Complex Designs
- **Any CFB Can Be Either 24V10 Logic or SRAM Block**
  - — Up to 40 Complex Macrocells
  - — 128 x 10 SRAM Configuration
  - — CFB Selectable 3.3V or 5V Outputs
  - — Open-Drain Output Option
- **24V10 Macrocell Features**
  - — Dual Feedback on All I/O Pins
  - — Allocation Supports up to 16 Product Terms per Macrocell with No Performance Penalty
  - — 12 Clocking Options
  - — Flexible Preset/Clear Options
  - — Selectable D/T Flip-Flops
  - — Fast 12-Bit Identity Compare Option
- **Supported by Industry Standard Design and Programming Tools**



290460-1



290460-2

Package Options

Pins	Package	Macrocells	I/O	Inputs	Clocks	JTAG/V_{PP}	V_{CC}	GND
44	PLCC	40	30	0	2	5	3	4
68	PLCC	40	40	10	2	5	5	6

Refer to the 1994 Programmable Logic Handbook for the complete data sheet on this device.

November 1993
Order Number: 290460-003

intel®

AP-366

APPLICATION NOTE

89C124FX Data/FAX Modem Chip Set

Reduction of Power Consumption

4

JIN LIEN LIN
TECHNICAL MARKETING ENGINEER

November 1992

Order Number: 292101-001

89C124FX Data/FAX Modem Chip Set
Reduction of Power Consumption

CONTENTS

	PAGE
INTRODUCTION	4-117
GENERAL DESCRIPTION	4-117
POWERDOWN DETECTION	4-117
BLOCKING DC PATH	4-117
CURRENT DRIFT	4-117
DESIGN TRADE OFF	4-117

INTRODUCTION

The 89C124FX Data/Fax Modem Chip Set Application Note provides the end user with applications and layout guidelines to reduce power consumption to a minimum when in the Power Down Mode.

GENERAL DESCRIPTION

When the 89C124FX is in the power down mode, the microcontroller (89C126FX) consumes very little power (less than 0.5 mW). However, the external memory, voltage regulators and peripherals draw excess current that makes the overall system power consumption more than 80 mW. This application note describes a method to reduce overall system power consumption to less than 1 mW when the 89C124FX is in the power down mode. Adding a power down feature in the microcontroller and reducing power sink to a minimum accomplishes this goal.

Three steps are required to reduce the overall system power consumption in the power down mode:

1. Detect power down in the microcontroller and isolate the power source from potential current sink from other components.
2. Inhibit the DC path from the power supply, through other components, to ground when the microcontroller is powered down.
3. Solve current drift problems due to floating inputs when power is removed from peripherals.

POWER DOWN DETECTION

Monitoring the clock output (CLKOUT, pin 65) from the microcontroller detects the power down condition. The CLKOUT pin is held high when the controller is in the power down mode. When power down is detected, the detector shuts off the +5V supply to all components except the microcontroller, RAM, and logic gates. The detector also shuts off power to the voltage regulators.

BLOCKING DC PATH

When the power down detector shuts off the bipolar switches, some of the device input pins become current sinks and drain current from the controller output pins when the output pins are high. These controller output pins are the CLKOUT and SCLK outputs. To eliminate these DC paths in the power down mode, add an inverter from the controller clock output (CLKOUT) to the AFE clock input (CLKIN) and use an AND gate to change the SCLK output to the AFE to a low.

CURRENT DRIFT

When the controller is in the power down mode, the SDATA pin becomes a floating input that can draw current in excess of 300 μA. Placing a 510 KΩ resistor between SDATA pin and ground solves this problem. Using a 100 KΩ resistor or lower may impede circuit functionality.

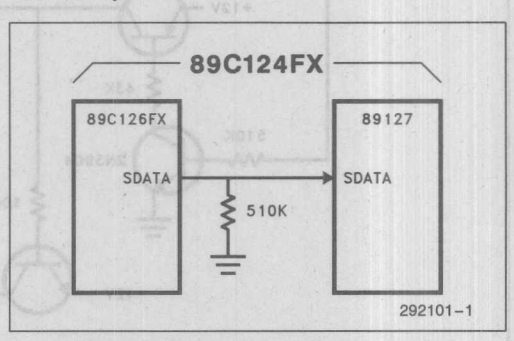


Figure 1. Placement of Current Drain Blocking Resistor

DESIGN TRADE OFF

The reduced power drain feature design trade-off, besides adding circuit complexity, is an additional 20 mW power consumption when the 89C124FX is active.

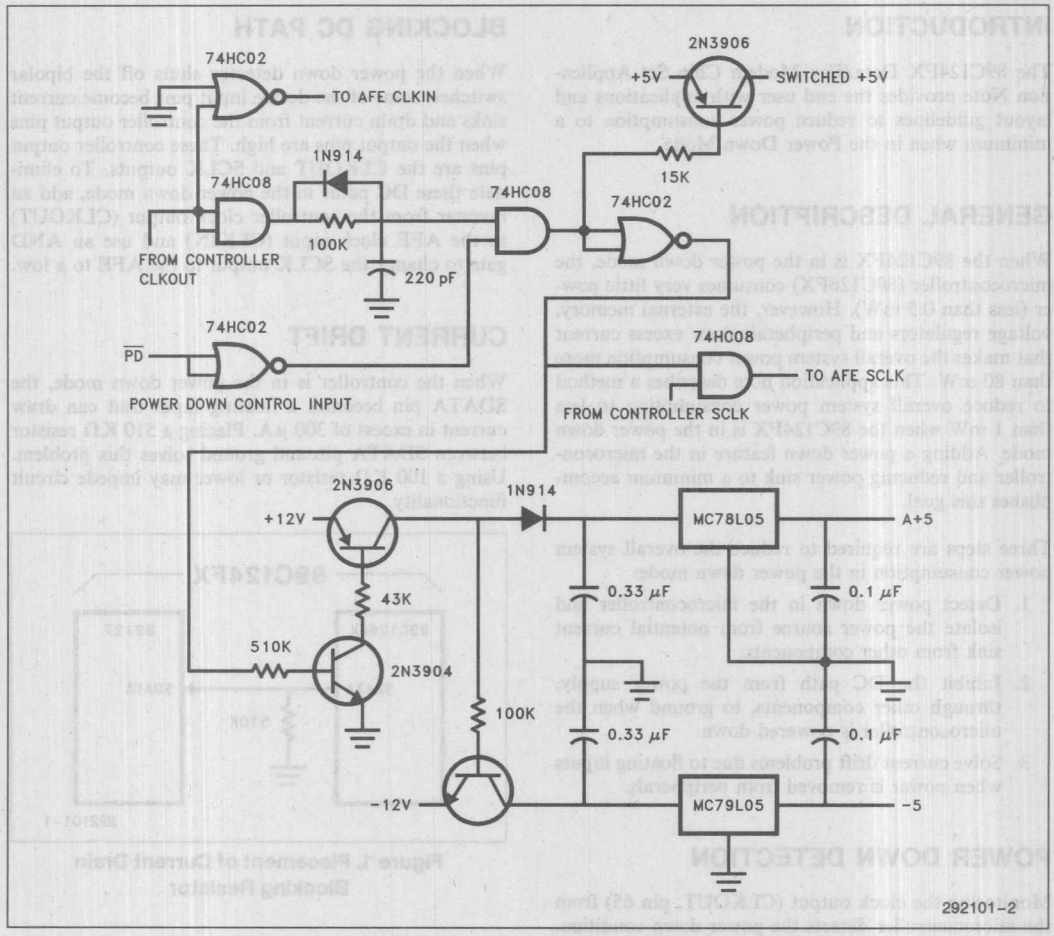


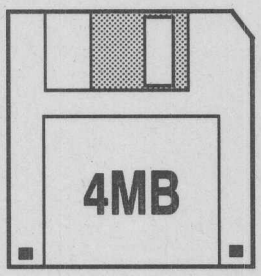
Figure 2. 89C124FX Power Consumption Circuit Modifications

intel®

AP-358

APPLICATION NOTE

Intel 82077SL for Super Dense Floppies



292093-1

4

KATEN A. SHAH
APPLICATION ENGINEER

September 1992

Order Number: 292093-002

Intel 82077SL for Super Dense Floppies

CONTENTS PAGE

INTRODUCTION 4-121

PURPOSE 4-121

PERPENDICULAR RECORDING MODE 4-121

PERPENDICULAR DRIVE FORMAT AND SPECIFICATION 4-123

PERPENDICULAR MODE COMMAND 4-123

CONTENTS PAGE

82077AA/SL'S PERPENDICULAR MODE SUPPORT 4-124

PROGRAMMING PERPENDICULAR MODE 4-126

INTERFACE BETWEEN 82077AA/SL AND THE DRIVE 4-129

82077SL 4 MB DESIGN 4-134

INTRODUCTION

The evolution of the floppy has been marked in little over a decade by a significant increase in capacity accompanied by a noticeable decrease in the form factor from the early 8 inch floppy disks to the present day 3.5 inch floppy disks. This decade will also be remarkable as OEMs adopt "Super" dense floppies.

The most commonly seen floppies today are invariably one of the form factors – the 5.25″ or the 3.5″. Each form factor has several associated capacity ranges. The 5.25″ floppies available are: 180 KB (single density), 360 KB (double density) and 1.2 MB (high density). The 3.5″ floppies available are: 720 KB (double density) and 1.44 MB (high density). The emerging super dense floppies will evolve on the installed base of 3.5″ floppies. The latest member of this set is the 2.88 MB (extra density) floppy, pioneered by Toshiba. The cornerstone of market acceptance of newer drives is compatibility to the older family. The 2.88 MB (formatted) floppy drive allows the user to format, read from and write to the lower density diskettes.

As programs and data files get bigger, the demand for higher capacity floppies becomes obvious. There are several 3.5″ higher density drives available from various vendors with capacities well into the 20 MB range. NEC has introduced a 13 MB drive and companies such as Insite have introduced 20 MB drives. Both drives require servo-mechanisms to accurately position the head over the right track. NEC's drive has the standard floppy drive interface whereas Insite's interface is SCSI based. The market for these floppy drives will remain a niche unless they receive more OEM support.

Initiated by Toshiba's research and innovation of the higher density 4 MB floppy disk media, the market is headed towards the super dense floppy drive. After IBM's endorsement of the 4 MB (unformatted) floppy disk drives on their PS/2 model 57 and PS/2 model 90, several OEMs have shown a growing interest in "super" dense floppy disk drives. The latest DOS 5.0 supports the new 4 MB floppy media and BIOS vendors like Pheonix, AMI, Award, Quadtel, System Soft, and Microid all support the newer 4 MB floppy media.

PURPOSE

An important consideration to implement the 4 MB floppy drive is the floppy disk controller. Intel's highly integrated floppy disk controller, 82077AA/SL, has led the market in supporting the 4 MB floppy drive. Two ingredients are necessary to fully support these drives: 1 Mbps transfer rate and the perpendicular recording mode. This paper deals with a discussion of what the perpendicular mode is and how can a 4 MB floppy disk drive be implemented in a system using the 82077AA/SL.

PERPENDICULAR RECORDING MODE

Toshiba has taken the 2 MB floppy and doubled the storage capacity by doubling the number of bits per track. Toshiba achieved this by an innovative magnetic recording mode, called the vertical or the perpendicular recording mode. This mode utilizes magnetization perpendicular to the recording medium plane. This is in contrast to the current mode of longitudinal recording which uses the magnetization parallel to the recording plane. By making the bits stand vertical as opposed to on their side, recording density is effectively doubled, Figure 1. The new perpendicular mode of recording not only produces sharp magnetization transitions necessary at higher recording densities, but is also more stable.

The 4 MB disks utilize barium ferrite coated substrates to achieve perpendicular mode of magnetization. Current disks use cobalt iron oxide (Co-g-Fe_2O_3) coating for longitudinal recording. The barium ferrite ensures good head to medium contact, stable output and durability in terms of long use. High coercivity is required to attain high recording density for a longitudinal recording medium (coercivity specification of a disk refers to the magnetic field strength required to make an accurate record on the disk). A conventional head could not be used in this case; however, the barium ferrite disk has low coercivity and the conventional ferrite head can be used. The new combination heads include a pre-erase mechanism, i.e., the ferrite ring heads containing erase elements followed by the read/write head. These erase elements have deep overwrite penetration and ensure complete erasure for writing new data. The distance between the erase elements and the read/write head is about 200mm. This distance is important from the floppy disk controller point of view and will be discussed in later sections.

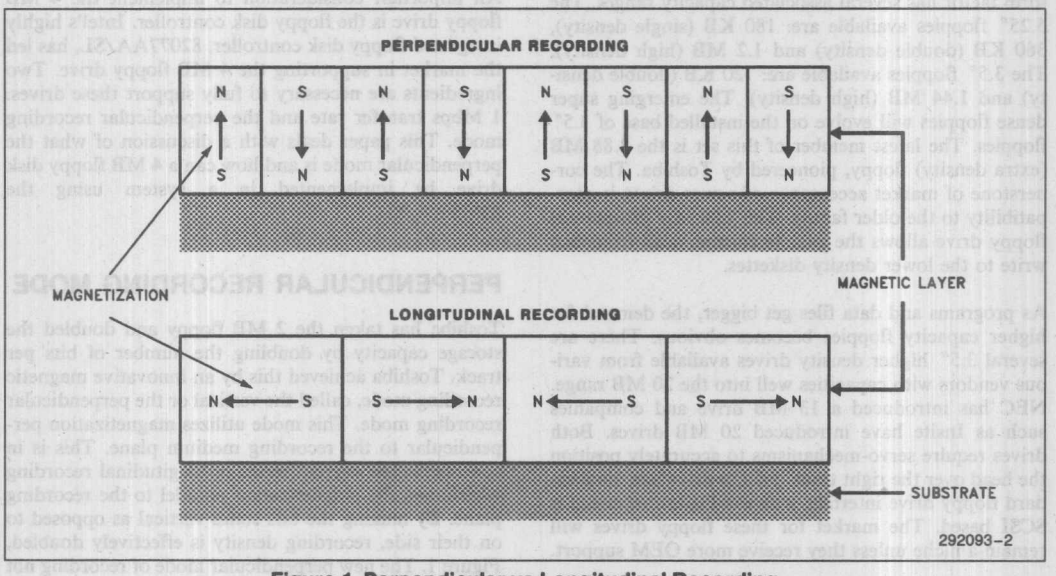


Figure 1. Perpendicular vs Longitudinal Recording

PERPENDICULAR DRIVE FORMAT AND SPECIFICATION

Figures 2a and 2b show the IBM drive format for both double density and perpendicular modes of recording. The main difference in recording format is the length of Gap2 between the ID field and the Data field. The main reason for the increased Gap2 length is the pre-erase head preceding the read/write head on the newer 4 MB floppy drives. The size of the data field is maintained at 512 KBytes standard. The increase in the capacity is implemented by increasing the number of sectors from 18 to 36. Table 1 shows the specifications of the various capacity 3.5" drives.

PERPENDICULAR MODE COMMAND

The current 82077AA/SL parts contain the "enhanced" perpendicular mode command as shown in Figure 3. This is a two byte command with the first byte being the command code (0x12H). The 2nd byte contains the parameters required to enable perpendicular mode recording. The former command (in the older 82077 parts) included only the WGATE and GAP bits. This command is compatible to the older mode where only the two LSBs are written. The enhanced mode allows system designers to designate specific drives as perpendicular recording drives. The second byte will be referenced as the PR[0:7] byte for ease of discussion. The following discusses the use of the enhanced perpendicular recording mode.

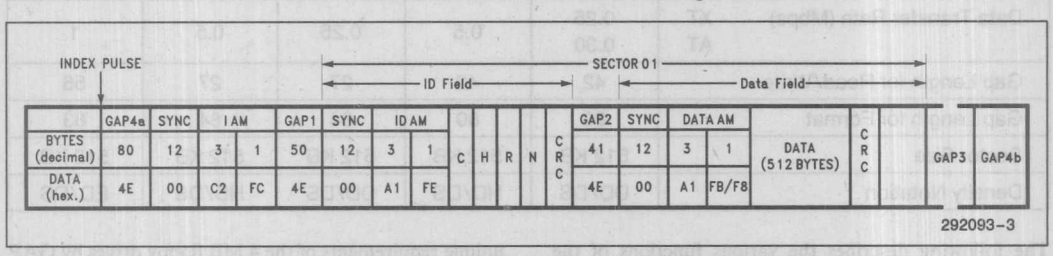


Figure 2a. Conventional IBM 1 MB and 2 MB Format (MFM)

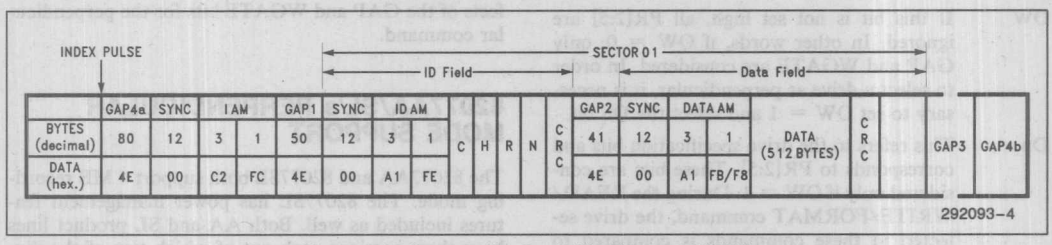


Figure 2b. Perpendicular 4 MB Format (MFM)

Phase	R/W	Data Bus D7	D6	D5	D4	D3	D2	D1	D0	Remarks
PERPENDICULAR MODE COMMAND										
Command	W	0	0	0	1	0	0	1	0	Command Code
	W	OW	0	D3	D2	D1	D0	GAP	WGATE	PR

Figure 3. Perpendicular Mode Command

Table 1. Specifications of FDDs

Various Parameters Used in the Different Kinds of FDDs.		5.25″ 360 KB	5.25″ 1.2 MB	3.5″ 720 KB	3.5″ 1.44 MB	3.5″ 2.88 MB
Number of Cylinders		40	80	80	80	80
Sectors/Track		9	15	9	18	36
Formatted Capacity		354 KB	1.2 MB	720 KB	1.44 MB	2.88MB
Unformatted Capacity		360 KB	1.6 MB	1 MB	2 MB	4 MB
Rotation Speed (rpm)	XT AT	300 360	360	300	300	300
Track Density (tpi)		48	96	135	135	135
Recording Density (bpi)		5876	9870	8717	17432	34868
Data Transfer Rate (Mbps)	XT AT	0.25 0.30	0.5	0.25	0.5	1
Gap Length for Read/Write		42	42	27	27	56
Gap Length for Format		80	80	84	84	83
Sector Size		512 KB	512 KB	512 KB	512 KB	512 KB
Density Notation		DD/DS	HD/DS	DD/DS	HD/DS	ED/DS

The following describes the various functions of the programmed bits in the PR:

OW — If this bit is not set high, all PR[2:5] are ignored. In other words, if OW = 0, only GAP and WGATE are considered. In order to select a drive as perpendicular, it is necessary to set OW = 1 and select the Dn bit.

Dn — This refers to the drive specification bits and corresponds to PR[2:5]. These bits are considered only if OW = 1. During the READ/WRITE/FORMAT command, the drive selected in these commands is compared to Dn. If the bits match then perpendicular mode will be enabled for that drive. For example, if D0 is set then drive 0 will be configured for perpendicular mode.

GAP — This alters the Gap2 length as required by the perpendicular mode format.

WGATE — Write gate alters timing of WE to allow for pre-erase loads in perpendicular drives.

The VCOEN timing and the length of the Gap2 field (explained above) can be altered to accommodate the unique requirements of the 4 MB floppy drives by GAP and WGATE bits of the PR. Table 2 describes the effects of the GAP and WGATE bits for the perpendicular command.

82077AA/SL's PERPENDICULAR MODE SUPPORT

The 82077AA and 82077SL both support 4 MB recording mode. The 82077SL has power management features included as well. Both AA and SL product lines have three versions each out of which two of the versions support the 4 MB floppy drives. The 82077AA-1, 82077AA, 82077SL, and 82077SL-1 all support the 4 MB floppy drives. A single command puts the 82077AA/SL into the perpendicular mode. This mode also requires the data rate to be set at 1 Mbps. The FIFO that is unique to Intel's 82077AA/SL parts may become necessary to remove the host interface bottleneck due to the higher data rate. The 4 MB floppy disk drives are downward compatible to 1 MB and 2 MB floppy diskettes. The following discussion explains the implications of the new 4 MB combination head and the functionality of the perpendicular mode command.

Table 2. Effects of GAP and WGATE Bits

GAP	WGATE	Mode	VCO Low Time after Index Pulse	Length of Gap2 Format Field	Portion of Gap2 Written by Write Data Operation	Gap2 VCO Low Time for Read Operations
0	0	Conventional	33 Bytes	22 Bytes	0 Bytes	24 Bytes
0	1	Perpendicular (Data Rate = 500 kbps)	33 Bytes	22 Bytes	19 Bytes	24 Bytes
1	0	Conventional	33 Bytes	22 Bytes	0 Bytes	24 Bytes
1	1	Perpendicular	33 Bytes	41 Bytes	38 Bytes	43 Bytes

The implementation of 4 MB drives requires understanding the Gap2 (see Figures 2a and 2b) and VCO timing requirements unique to these drives. These new requirements are dictated by the design of the "combination head" in these drives. Rewriting of disks in the 4 MB drives requires a pre-erase gap to erase the magnetic flux on the disk preceding the writing by the read/write gap. The read/write gap in the 4 MB drive does not have sufficient penetration (as shown in Figure 4a) to overwrite the existing data. In the conventional drives, the read/write gap had sufficient depth and could effectively overwrite the older data as depicted in Figure 4b. It must be noted that it is necessary to write the conventional 2 MB media in the 4 MB drive at 500 Kbps perpendicular mode. This ensures proper erasure of existing data and reliable write of the new data. The pre-erase gap in the 4 MB floppy drives is activated only during format and write commands. Both the pre-erase gap and read/write gap are activated at the same time.

As shown in Figure 4a, the pre-erase gap precedes the read/write gap by 200mm. This distance translated to bytes is about 38 bytes at a data rate of 1 Mbps and 19 bytes at 500 Kbps. Whenever the read/write gap is enabled by the Write Gate signal the pre-erase gap is activated at the same time.

4

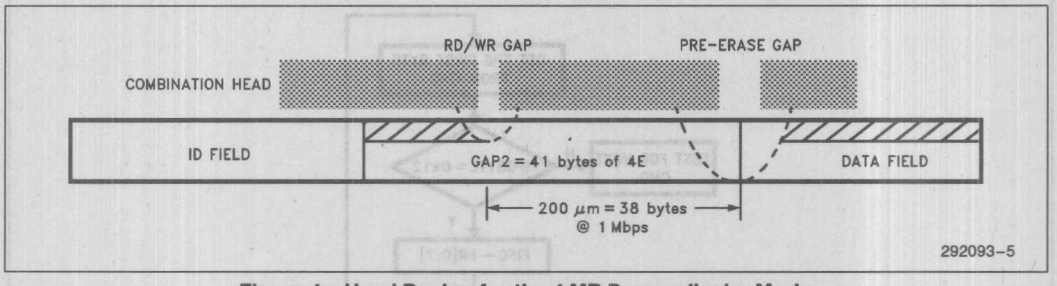


Figure 4a. Head Design for the 4 MB Perpendicular Mode

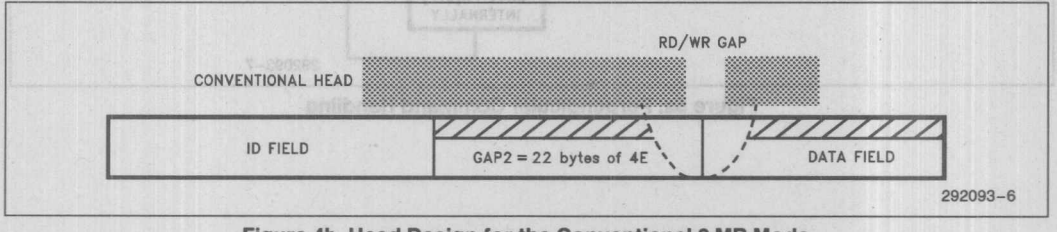


Figure 4b. Head Design for the Conventional 2 MB Mode

In conventional drives, the Write Gate is asserted at the beginning of the sync field, i.e., when the read/write is at the beginning of the data field. The controller then writes the new sync field, data address mark, data field and CRC (see Figure 2a). With the combination head, the read/write gap must be activated in the Gap2 field to ensure proper write of the new sync field. To accommodate both the distance between the pre-erase gap and read/write gap and the head activation and deactivation time, the Gap2 field is expanded to a length of 41 bytes at 1 Mbps (see Figure 2b). Since the bit density is proportional to the data rate, 19 bytes will be written in the Gap2 field at 500 Kbps data rate in the perpendicular mode.

On the read back by the 82077AA/SL, the controller must begin the synchronization at the beginning of the sync field. For conventional mode, the internal PLL VCO is enabled (VCOEN) approximately 24 bytes from the start of the Gap2 field. However, at 1 Mbps perpendicular mode the VCOEN goes active after 43 bytes to accomodate the increased Gap2 field size. For each case, a 2 byte cushion is maintained from the beginning of the sync field to avoid write splices caused by motor speed variation.

It should be noted that none of the alterations in Gap2 size, VCO timing or Write Gate timing affect the normal program flow. Once the perpendicular command is invoked, 82077AA/SL behaviour from the user standpoint is unchanged.

PROGRAMMING PERPENDICULAR MODE

Figures 5a and 5b show a flowchart on how the perpendicular recording mode is implemented on the 82077AA/SL. The perpendicular mode command can be issued during initialization. As shown in Figure 5a the perpendicular command stores the PR value internally. This value is used during the data transfer commands for configuration in order to deal with the perpendicular drives. Table 2 shows how the Gap2 length, VCOEN timing or Write Gate timing is affected. The OW bit is also tested for in this part of the loop. The enhanced perpendicular mode is enabled by setting the OW = 1, setting the Dn bits corresponding to the installed perpendicular drive high and leaving PR[0:1] = '00'.

As shown in Figure 5b, the Gap2 length is initially set to the conventional length of 22 bytes. Next the PR[0:1] bits (GAP, WGATE) are checked if they are set to '00'. If the PR[0:1] bits are set to '10' then, perpendicular mode is disabled and conventional mode is retained. If the PR[0:1] = '01' or '11' the VCOEN is

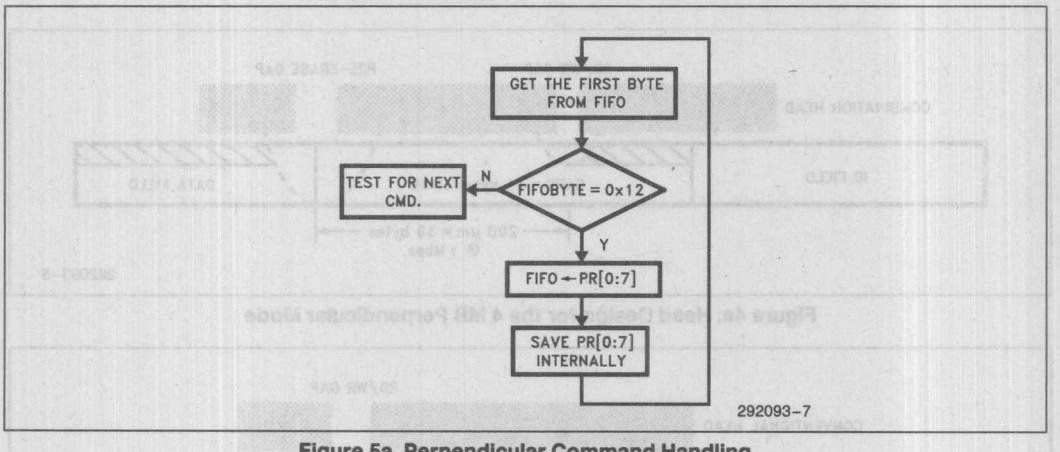


Figure 5a. Perpendicular Command Handling

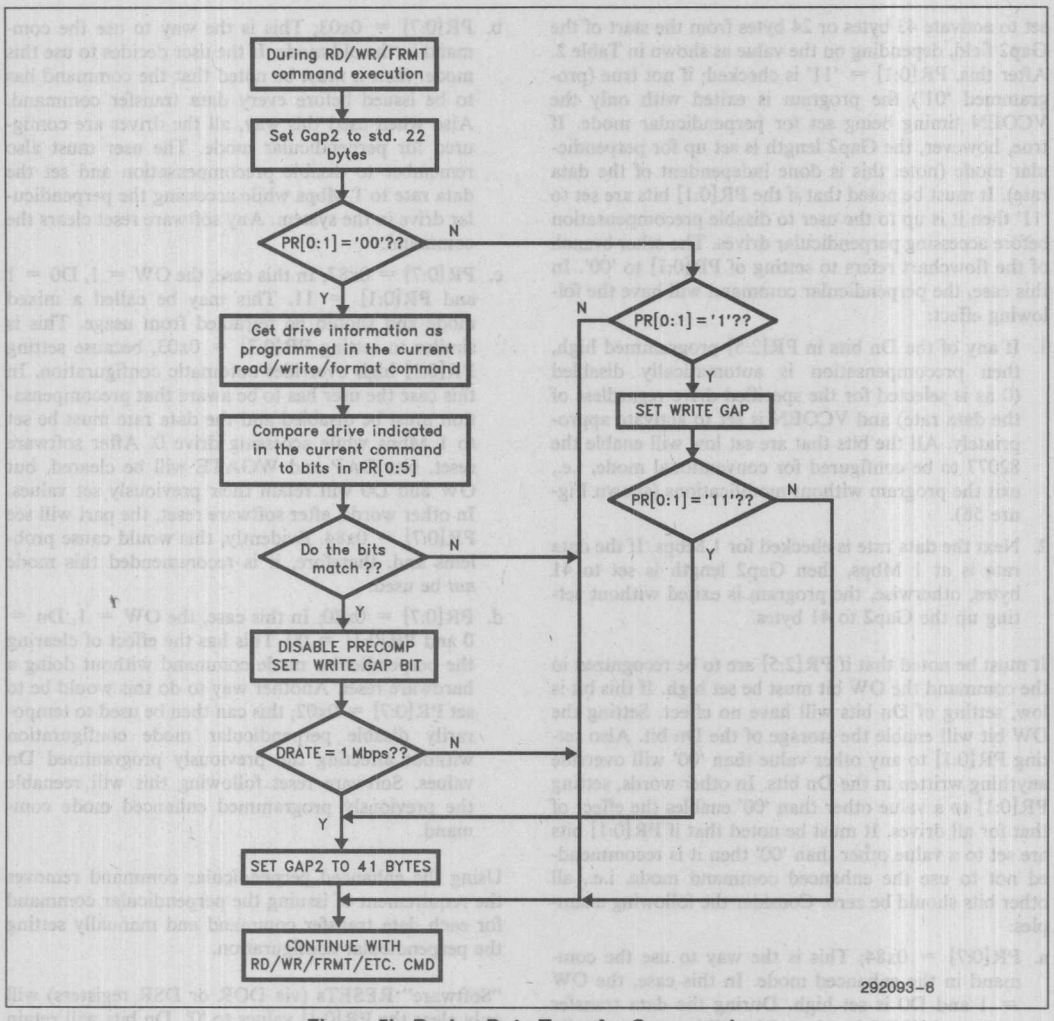


Figure 5b. During Data Transfer Commands

4

set to activate 43 bytes or 24 bytes from the start of the Gap2 field, depending on the value as shown in Table 2. After this, PR[0:1] = '11' is checked; if not true (programmed '01') the program is exited with only the VCOEN timing being set for perpendicular mode. If true, however, the Gap2 length is set up for perpendicular mode (note: this is done independent of the data rate). It must be noted that if the PR[0:1] bits are set to '11' then it is up to the user to disable precompensation before accessing perpendicular drives. The other branch of the flowchart refers to setting of PR[0:1] to '00'. In this case, the perpendicular command will have the following effect:

1. If any of the Dn bits in PR[2:5] programmed high, then precompensation is automatically disabled (0 ns is selected for the specified drive regardless of the data rate) and VCOEN is set to activate appropriately. All the bits that are set low will enable the 82077 to be configured for conventional mode, i.e., exit the program without modifications (shown Figure 5b).
2. Next the data rate is checked for 1 Mbps. If the data rate is at 1 Mbps, then Gap2 length is set to 41 bytes, otherwise, the program is exited without setting up the Gap2 to 41 bytes.

It must be noted that if PR[2:5] are to be recognized in the command the OW bit must be set high. If this bit is low, setting of Dn bits will have no effect. Setting the OW bit will enable the storage of the Dn bit. Also setting PR[0:1] to any other value than '00' will override anything written in the Dn bits. In other words, setting PR[0:1] to a value other than '00' enables the effect of that for all drives. It must be noted that if PR[0:1] bits are set to a value other than '00' then it is recommended not to use the enhanced command mode, i.e., all other bits should be zero. Consider the following examples:

a. PR[0:7] = 0x84; This is the way to use the command in the enhanced mode. In this case, the OW = 1 and D0 is set high. During the data transfer command, if D0 is selected it will be automatically configured for perpendicular mode. If D1 is accessed, however, it will be configured for conventional mode. Similarly, if PR[0:7] = 0x88 then D1 is configured for perpendicular mode and D0 is configured for conventional mode. Software resets do not clear this mode.

b. PR[0:7] = 0x03; This is the way to use the command in the old mode. If the user decides to use this mode, then it must be noted that the command has to be issued before every data transfer command. Also when used this way, all the drives are configured for perpendicular mode. The user must also remember to disable precompensation and set the data rate to 1 Mbps while accessing the perpendicular drive in the system. Any software reset clears the command.

c. PR[0:7] = 0x87; In this case, the OW = 1, D0 = 1 and PR[0:1] = 11. This may be called a mixed mode and should be refrained from usage. This is similar to setting PR[0:7] = 0x03, because setting PR[0:1] high overrides automatic configuration. In this case the user has to be aware that precompensation must be disabled and the data rate must be set to 1 Mbps while accessing drive 0. After software reset, bits GAP and WGATE will be cleared, but OW and D0 will retain their previously set values. In other words, after software reset, the part will see PR[0:7] = 0x84. Evidently, this would cause problems and, therefore, it is recommended this mode *not* be used.

d. PR[0:7] = 0x80; In this case, the OW = 1, Dn = 0 and PR[0:1] = 00. This has the effect of clearing the perpendicular mode command without doing a hardware reset. Another way to do this would be to set PR[0:7] = 0x02; this can then be used to temporarily disable perpendicular mode configuration without affecting the previously programmed Dn values. Software reset following this will reenable the previously programmed enhanced mode command.

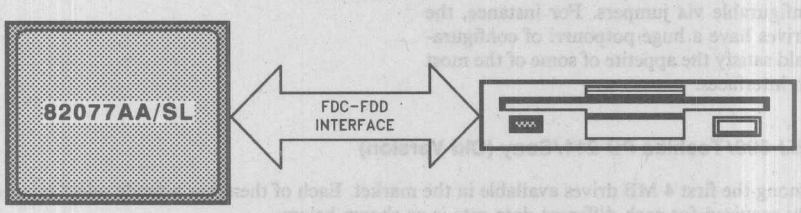
Using the enhanced perpendicular command removes the requirement of issuing the perpendicular command for each data transfer command and manually setting the perpendicular configuration.

"Software" RESETs (via DOR or DSR registers) will only clear the PR[0:1] values to '0'. Dn bits will retain their previously programmed values. "Hardware" RESETs will clear all the programmed bits including OW and Dn bits to '0'. The status of these bits can be determined by issuing the dumpreg command and checking the 8th result byte. This byte will contain the programmed values of the Dn and PR[0:1] bits as shown in Figure 6. The OW bit is *not* returned in this result byte.

Phase	R/W	Data Bus								Remarks
		D7	D6	D5	D4	D3	D2	D1	D0	
DUMPREG COMMAND										
Command	R	Eighth Result Byte								
		LOCK	0	D3	D2	D1	D0	GAP	WGATE	

Figure 6. Dumpreg Command

INTERFACE BETWEEN 82077AA/SL AND THE DRIVE



There is currently no industry-wide standard for the FDC to FDD interface. There are numerous floppy drive vendors, each with their own modes and interface pins to enable 4 MB perpendicular mode. The drive interface not only varies from manufacturer to manufacturer but also within a manufacturer's product line. The differences on the interface mainly originate from configuring the floppy drive into the 4 MB mode. Depending on the drive, the differences can create problems of daisy-chaining a 4 MB drive with the standard 1 MB and 2 MB drives. Of course, for laptops this is not a problem since most of them use a single floppy drive. Lack of an industry standard makes it necessary to look at each drive and build a interface for that particular drive.

The following is a brief discussion about some of the floppy drives available in the market and how these can be interfaced with the 82077AA/SL. It is important to note that although a manufacturer's name may be given in connection with the interface described, Intel does not guarantee that the interface discussed will apply to all the drives from that manufacturer. The main goal is introduce to the reader how to interface the 82077AA/SL with a 4 MB floppy drive.

Previously, for the conventional 1 MB and 2 MB AT mode drives, a single Density Select input was used by floppy drives to select between high density and low density drives. A high on this input enabled high density operation (500 Kbps) whereas a low enabled low density operation (300 Kbps/250 Kbps). This signal was asserted high or low by the floppy disk controller depending on the data rate programmed. For the 4 MB operation, there are two inputs defined by the floppy drive manufacturers. The polarity of these inputs enables the selected density operation. Implementing this requires at least 1 new pin to be defined on the FDC-FDD interface. Most floppy vendors have elected to take pin 2 (originally density select) and redefine the polarity to conform to one of these new density select inputs and another pin to be the other density select input. However, the new density select on pin 2 is not compatible to the old density select input in many of the floppy drives. This precludes the user from daisy chaining 4 MB drives with conventional drives. Another problem is that the second density select pin varies on its location on the FDC-FDD interface from drive to drive.

The way that the BIOS determines what type of diskette is in what type of drive is by trial and error. The system tries to read the diskette at 250 Kbps; if it fails then it will set the data rate to higher value and retry. The BIOS does this until the right data rate is selected. This method will still be implemented for the 4 MB drives by some BIOS vendors. However, the 4 MB drives available today also have two media sense ID pins that relate to the user what type of media is present in the floppy drive. This information will also require two pins on the FDC-FDD interface. The location of these pins is once again variable from drive to drive.

dardization problem by including an auto configuration in the drive. In these cases, the type of floppy put into the drive is sensed by the hole (each 4/2/1 MB diskette has a hole in different locations identifying it) on the diskette. Then the drive automatically sets itself up for this mode. The BIOS must obviously set up the floppy disk controller for the correct data rate which could be done if the media sense ID was read and decoded as to the data rate. Due to lack of extra pins on the even side of the floppy connector the newer locations of some of the functions are migrating to the odd pins (previously all grounded). Some drive manufacturers have even made this configurable via jumpers. For instance, the new TEAC drives have a huge potpourri of configurations that would satisfy the appetite of some of the most finicky system interfaces.

DRATE0 and DRATE1 (pins 28 and 29 respectively) which directly reflect the data rate programmed in the DSR and CCR registers. These two pins can be used to select the correct density on the drive. These two can also be used with the combination of DENSEL to select the correct data rate. At the present time the 82077AA/SL does not support media sense ID. However, the user could easily make it readable directly by BIOS. The following is a discussion on what combination of DRATE0, DRATE1, and DENSEL could be used to interface to some of the currently available floppy drives.

1. TEAC 235J-600/Toshiba PD-211/Sony (Old Version)

These were among the first 4 MB drives available in the market. Each of them has a mode select input on pins 2 and 6. The polarity required for each different data rate is as shown below:

Data Rate	Capacity	DRATE1	DRATE0	MODSEL0 pin 2	MODSEL1 pin 6
1 Mbps	4 MB	1	1	1	0
500 Kbps	2 MB	0	0	0	1
300 Kbps/ 1 Mbps	4 MB	0	1	1	1
250 Kbps	1 MB	1	0	0	0

It is clear from the above that DRATE0 = MODSEL0 and MODSEL1 = DRATE1#. This would mean taking the drate signals onto pins 2 and 6 of the FDC-FDD interface. Unfortunately this solution requires an inverting gate. TEAC has recently, however, come out with a new version called TEAC 235J-3653. On this drive there are a number of possible configurations into which the drive can be put into, however, only the best way to interface to the 82077AA/SL will be discussed. The requirements are as shown below. This shows that HDIN = DENSEL (original signal for conventional drives) and EDIN = DRATE0. As suggested in the TEAC spec for method 1, the straps connected are MSC, HI2 (sets HDIN on pin 2), DC34 and EI6 (sets EDIN on pin 6). Pins 4, 29, and 33 are left open. Since pin 2 has the same polarity as the conventional drive requirement and the secondary input is connected via pin 6 (no connect on the conventional drives) daisy chaining this TEAC drive with a conventional drive does not cause any incompatibility. Figure 7 shows how the TEAC can be connected to the 82077AA/SL. It also shows daisy chaining of the TEAC drive with a conventional drive.

Data Rate	Capacity	DENSEL	DRATE1	DRATE0	HDIN pin 2	EDIN pin 6
1 Mbps	4 MB	1	1	1	X	1
500 Kbps	2 MB	1	0	0	1	0
300 Kbps/ 1 Mbps	4 MB	0	0	1	X	1
250 Kbps	1 MB	0	1	0	0	0

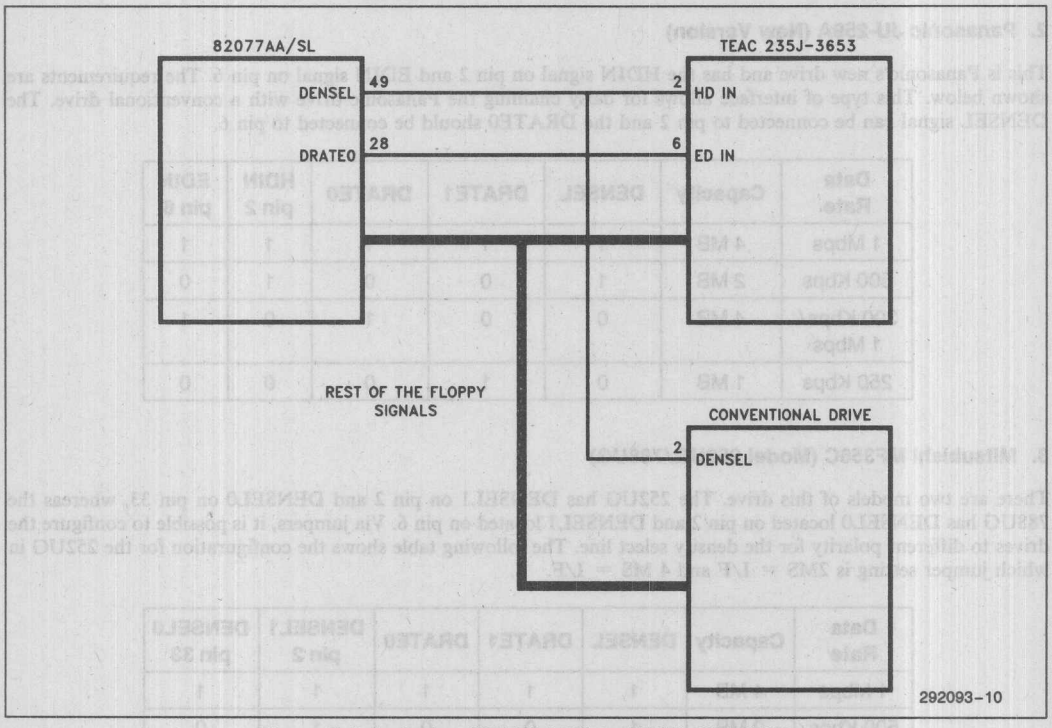


Figure 7. Interfacing 82077AA/SL to TEAC 235J-3653

2. Panasonic JU-259A (New Version)

This is Panasonic's new drive and has the HDIN signal on pin 2 and EDIN signal on pin 6. The requirements are shown below. This type of interface allows for daisy chaining the Panasonic drive with a conventional drive. The DENSEL signal can be connected to pin 2 and the DRATE0 should be connected to pin 6.

Data Rate	Capacity	DENSEL	DRATE1	DRATE0	HDIN pin 2	EDIN pin 6
1 Mbps	4 MB	1	1	1	1	1
500 Kbps	2 MB	1	0	0	1	0
300 Kbps/ 1 Mbps	4 MB	0	0	1	0	1
250 Kbps	1 MB	0	1	0	0	0

3. Mitsubishi MF356C (Model 252UG/788UG)

There are two models of this drive. The 252UG has DENSEL1 on pin 2 and DENSEL0 on pin 33, whereas the 788UG has DENSEL0 located on pin 2 and DENSEL1 located on pin 6. Via jumpers, it is possible to configure the drives to different polarity for the density select line. The following table shows the configuration for the 252UG in which jumper setting is 2MS = I/F and 4 MS = I/F.

Data Rate	Capacity	DENSEL	DRATE1	DRATE0	DENSEL1 pin 2	DENSEL0 pin 33
1 Mbps	4 MB	1	1	1	1	1
500 Kbps	2 MB	1	0	0	1	0
300 Kbps/ 1 Mbps	4 MB	0	0	1	0	1
250 Kbps	1 MB	0	1	0	0	0

The correct connection requirement is: DENSEL (from 82077AA/SL) = DENSEL1 and DRATE0 = DENSEL0. Although there are other configurations, this provides the best one, since daisy chaining is possible without any problem.

4. Epson SMD-1060

This drive has 3 different modes of operation. Mode B is the best and is similar to Mitsubishi's drives as described above. In this mode, HDI signal is connected to pin 2 and EDI is connected to pin 33. Mode B is enabled by inserting jumpers across 3-4 and 7-8 (SS01 B block) and 1-2 and 3-4 (SS03 block) for the drive with the power separated type (i.e., a connector for the floppy signals and another one for power supply) of 34-pin connector.

Data Rate	Capacity	DENSEL	DRATE1	DRATE0	HDI pin 2	EDI pin 33
1 Mbps	4 MB	1	1	1	1	1
500 Kbps	2 MB	1	0	0	1	0
300 Kbps/ 1 Mbps	4 MB	0	0	1	0	1
250 Kbps	1 MB	0	1	0	0	0

As demonstrated by the table, HDI = DENSEL and EDI = DRATE0. These connections would ensure daisy chaining capability without any problems.

5. Sony MP-F40W-14/15

The dash 14 and 15 are two drives from Sony that handle 4 MB requirements. The MP-F40W-14 has the DENSITY SELECT 1, DENSITY SELECT 0 on pins 2 and 33 respectively, whereas the MP-F40W-15 has the DENSITY SELECT 1, DENSITY SELECT 0 on pins 2 and 6 respectively. As it is obvious from the table below, daisy chaining is easily done if the 82077AA/SL is connected in the PS/2 mode (by tying IDENT low) with either type of drive, the only difference being the location of DENSITY SELECT 0.

Data Rate	Capacity	DENSEL PS/2 mode (IDENT = 0)	DRATE1	DRATE0	DENSITY SELECT1 pin 2	DENSITY SELECT0 pin 6/33
1 Mbps	4 MB	0	1	1	0	1
500 Kbps	2 MB	0	0	0	0	0
300 Kbps/ 1 Mbps	4 MB	1	0	1	1	1
250 Kbps	1 MB	1	1	0	1	0

If the drive is used in the PS/2 mode, then DENSITY SELECT1 = DENSEL and DENSITY SELECT0 = DRATE0. To use the drive in AT mode, DENSITY SELECT1 = DRATE1 and DENSITY SELECT0 = DRATE0, as shown below. However, daisy chaining is not possible.

Data Rate	Capacity	DENSEL PS/2 mode (IDENT = 0)	DRATE1	DRATE0	DENSITY SELECT1 pin 2	DENSITY SELECT0 pin 6/33
1 Mbps	4 MB	0	1	1	1	1
500 Kbps	2 MB	0	0	0	0	0
300 Kbps/ 1 Mbps	4 MB	1	0	1	0	1
250 Kbps	1 MB	1	1	0	1	0

6. Toshiba ND3571

Toshiba MB drive has the HD mode selection on pin 6 and ED mode selection on pin 2. This causes daisy chaining problems with conventional drives as shown in the figure below:

Data Rate	Capacity	DENSEL	DRATE1	DRATE0	ED Mode pin 2	HD Mode pin 6
1 Mbps	4 MB	1	1	1	1	1
500 Kbps	2 MB	1	0	0	0	1
300 Kbps/ 1 Mbps	4 MB	0	0	1	1	0
250 Kbps	1 MB	0	1	0	0	0

The DENSEL from the 82077 is connected to pin 6 and DRATE0 is connected to pin 2.

82077SL 4 MB DESIGN

This section presents a design application of a PC/AT compatible floppy disk controller. The 82077SL integrates the entire PC/AT controller design with the exception of the address decode on a single chip. The schematic for this solution is shown in Figure 8. The chip select for the 82077SL is generated by a 85C220 μPLD that is programmed to decode addresses 03F0H through 03F7H when AEN is low. The programming equations for the μPLD is in the Intel's .ADF format and can be processed using the IPLSII compiler (available from Intel).

A floppy disk interface is provided by on-chip output buffers with a 40 mA sink capability. The outputs from the disk drive are terminated at the floppy disk controller with a 1 KΩ resistor pack. The 82077SL disk interface inputs contain a Schmitt trigger input structure for higher noise immunity. The host interface is a similar direct connection with on-chip 12 mA sink capable buffers on DB0–7, INT and DRQ.

The schematic shows eleven jumpers numbered J1 through J11. The table below describes the functions of these jumpers as well as their normal connections. The normal connections allow the BIOS to work without modification. In the normal mode, the 82077SL responds to DRQ2 and DACK2# as well as IRQ6. Depending on the type of drive interfaced to this board, the DENOUT0 and DENOUT1 signals can be tied. With the setting to 2–3 on J8 and J9, the default setting is DENSEL on DRVDEN0 and DRATE0 on DRVDEN1. PIN6/33 SELECT is used to set for pin 6 as the EDIN input. The J11 should always be closed. It can be used to measure the current consumption of 82077SL. J7 selects between the primary and secondary address spaces. There are two resistor packs used for pullups on input signals from the floppy drive interface. These resistors are rated at 1K. Please note that if using older 5.25″ drives, the pullup on some of them is 150Ω. Most modem 5.25″ drives use a 1K value. In order to ensure the correct value please refer to the floppy drive specification manual.

For further information, please contact your local Intel sales office.

Jumper	Description	Normal Connection
J1	DRQ1: DMA request 1 used with DACK1# to allow for DMA transfers	Open
J2	DRQ2: DMA request 2 used with DACK2# to allow for DMA transfers	Closed
J3	DACK1: DMA acknowledge 1 used with DRQ1 to allow for DMA transfers	Open
J4	DACK2: DMA acknowledge 2 used with DRQ2 to allow for DMA transfers	Closed
J5	IRQ5: Interrupt line 5 used to generate floppy interrupts	Open
J6	IRQ6: Interrupt line 6 used to generate floppy interrupts	Closed
J7	DRV2: Address selection (between 3FX and 37X address ranges)	Open
J8	DENOUT0: Used with DENOUT1 to select the values of DRVDEN1,0	2–3
J9	DENOUT1: Used with DENOUT0 to select the values of DRVDEN1,0	2–3
J10	PIN6/33 SELECT: Used to select between pin 6 and pin 33 for EDIN input	1–2 or 2–3
J11	V_{BB}/V_{CC}: Connection between two power layers	Closed

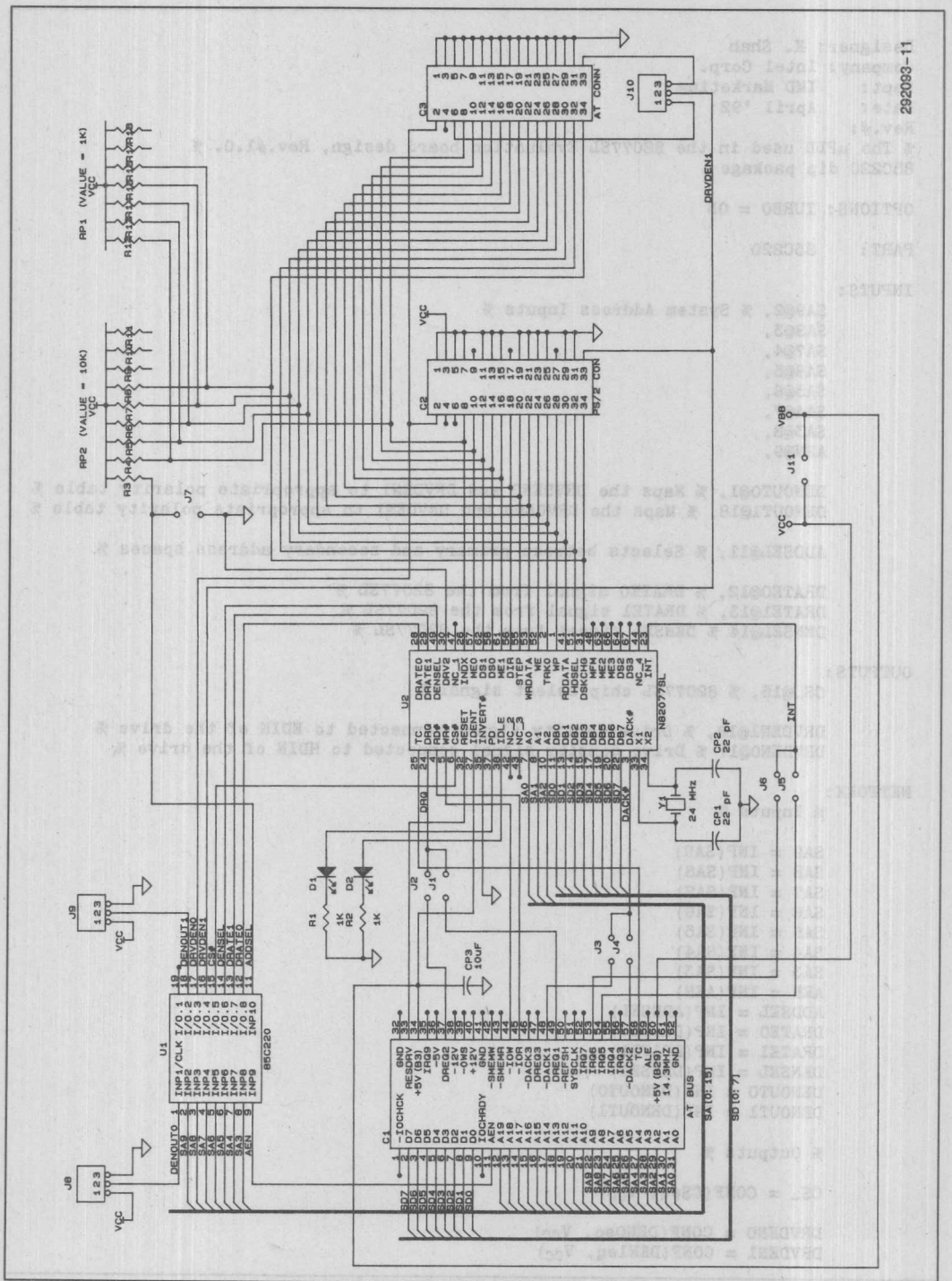


Figure 8. 82077SL Evaluation Board

```
Designer: K. Shah
Company: Intel Corp.
Dept:    IMD Marketing
Date:    April '92
Rev.#:
% The µPLD used in the 82077SL Evaluation board design, Rev.#1.0. %
85C220 dip package

OPTIONS: TURBO = ON

PART:    85C220

INPUTS:
        SA9@2, % System Address Inputs %
        SA8@3,
        SA7@4,
        SA6@5,
        SA5@6,
        SA4@7,
        SA3@8,
        AEN@9,

        DENOUT0@1, % Maps the DRVDEN0 and DRVDEN1 to appropriate polarity table %
        DENOUT1@18, % Maps the DRVDEN0 and DRVDEN1 to appropriate polarity table %

        ADDSEL@11, % Selects between primary and secondary address spaces %

        DRATE0@12, % DRATE0 signal from the 82077SL %
        DRATE1@13, % DRATE1 signal from the 82077SL %
        DENSEL@14 % DENSEL signal from the 82077SL %

OUTPUTS:
        CS_@15, % 82077SL chip select signal %

        DRVDEN1@16, % Drive density signal connected to EDIN of the drive %
        DRVDEN0@17 % Drive density signal connected to HDIN of the drive %

NETWORK:
        % Inputs %

        SA9 = INP(SA9)
        SA8 = INP(SA8)
        SA7 = INP(SA7)
        SA6 = INP(SA6)
        SA5 = INP(SA5)
        SA4 = INP(SA4)
        SA3 = INP(SA3)
        AEN = INP(AEN)
        ADDSEL = INP(ADDSEL)
        DRATE0 = INP(DRATE0)
        DRATE1 = INP(DRATE1)
        DENSEL = INP(DENSEL)
        DENOUT0 = INP(DENOUT0)
        DENOUT1 = INP(DENOUT1)

        % Outputs %

        CS_ = CONF(CSeq, VCC)

        DRVDEN0 = CONF(DEN0eq, VCC)
        DRVDEN1 = CONF(DEN1eq, VCC)
```

```
EQUATIONS:

        % CS_is activated for 3F0-3F7 and 370-377 address spaces %
        CSeq = (AEN' * SA9 * SA8 * SA7' * SA6 * SA5 * SA4 * SA3' * ADDSEL'
               + AEN' * SA9 * SA8 * SA7 * SA6 * SA5 * SA4 * SA3' * ADDSEL)';

        % These are the signals generated on DRVDEN0 and DRVDEN1 for the FDC-FDD
          interface
        DENOUT1  DENOUT0  DRVDEN0  DRVDEN1
             0     0      DENSEL   DRATE0
             0     1      DENSEL'  DRATE0
             1     0      DRATE1   DRATE0
             1     1      DRATE0   DRATE1
        %

        DEN0eq = DENSEL * (DENOUT0' * DENOUT1') + DENSEL' * (DENOUT0 * DENOUT1')
                + DRATE1 * (DENOUT0' * DENOUT1) + DRATE0 * (DENOUT0 * DENOUT1);
        DEN1eq = DRATE1 * (DENOUT0 * DENOUT1) + DRATE0 * (DENOUT0' + DENOUT1');

END$
```

82077SL Application Note Revision Summary

The following changes have been made since revision 001:

Table 2 kBps was corrected to kbps.

Page 4-132 3. Mitsubishi MF356C description modified to read: "There are two models of this drive. The 252UG has DENSEL1 on pin 2 and DENSEL0 on pin 33, whereas the 788UG has DENSEL0 located on pin 2 and DENSEL1 located on pin 6. Via jumpers, it is possible to configure the drives to different polarity for the density select lines. The following table shows the configuration for the 252UG in which jumper setting is 2 MS = I/F and 4 MS = I/F."

Figure 8 Arrow added to diagram.

Page 4-137 Columns corrected to line up properly.

Flash Memory Components 5

5 Flash Memory Components

28F001BX-T/28F001BX-B
1M (128K x 8) CMOS FLASH MEMORY

- **High Integration Blocked Architecture**
  - **One 8 KB Boot Block w/Lock Out**
  - **Two 4 KB Parameter Blocks**
  - **One 112 KB Main Block**
- **100,000 Erase/Program Cycles Per Block**
- **Simplified Program and Erase**
  - **Automated Algorithms via On-Chip Write State Machine (WSM)**
- **SRAM-Compatible Write Interface**
- **Deep-Powerdown Mode**
  - **0.05 μA I_{CC} Typical**
  - **0.8 μA I_{PP} Typical**
- **12.0V $\pm$5% V_{PP}**
- **High-Performance Read**
  - **120 ns Maximum Access Time**
  - **5.0V $\pm$10% V_{CC}**
- **Hardware Data Protection Feature**
  - **Erase/Write Lockout during Power Transitions**
- **Advanced Packaging, JEDEC Pinouts**
  - **32-Pin PDIP**
  - **32-Lead PLCC, TSOP**
- **ETOX II Nonvolatile Flash Technology**
  - **EPROM-Compatible Process Base**
  - **High-Volume Manufacturing Experience**
- **Extended Temperature Options**

Intel's 28F001BX-B and 28F001BX-T combine the cost-effectiveness of Intel standard flash memory with features that simplify write and allow block erase. These devices aid the system designer by combining the functions of several components into one, making boot block flash an innovative alternative to EPROM and EEPROM or battery-backed static RAM. Many new and existing designs can take advantage of the 28F001BX's integration of blocked architecture, automated electrical reprogramming, and standard processor interface.

The 28F001BX-B and 28F001BX-T are 1,048,576 bit nonvolatile memories organized as 131,072 bytes of 8 bits. They are offered in 32-pin plastic DIP, 32-lead PLCC and 32-lead TSOP packages. Pin assignment conform to JEDEC standards for byte-wide EPROMs. These devices use an integrated command port and state machine for simplified block erasure and byte reprogramming. The 28F001BX-T's block locations provide compatibility with microprocessors and microcontrollers that boot from high memory, such as Intel's MCS-186 family, 80286, i386™, i486™, i860™ and 80960CA. With exactly the same memory segmentation, the 28F001BX-B memory map is tailored for microprocessors and microcontrollers that boot from low memory, such as Intel's MCS-51, MCS-196, 80960KX and 80960SX families. All other features are identical, and unless otherwise noted, the term 28F001BX can refer to either device throughout the remainder of this document.

The boot block section includes a reprogramming write lock out feature to guarantee data integrity. It is designed to contain secure code which will bring up the system minimally and download code to the other locations of the 28F001BX. Intel's 28F001BX employs advanced CMOS circuitry for systems requiring high-performance access speeds, low power consumption, and immunity to noise. Its 120 ns access time provides no-WAIT-state performance for a wide range of microprocessors and microcontrollers. A deep-powerdown mode lowers power consumption to 0.25 μW typical through V_{CC}, crucial in laptop computer, handheld instrumentation and other low-power applications. The RP# power control input also provides absolute data protection during system powerup or power loss.

Manufactured on Intel's ETOX process base, the 28F001BX builds on years of EPROM experience to yield the highest levels of quality, reliability, and cost-effectiveness.

Refer to the 1994 Memory Products Handbook for the complete data sheet on this device.

October 1993
Order Number: 290406-005

28F200BX-T/B, 28F002BX-T/B 2-MBIT (128K x 16, 256K x 8) BOOT BLOCK FLASH MEMORY FAMILY

- x8/x16 Input/Output Architecture
  - 28F200BX-T, 28F200BX-B
  - For High Performance and High Integration 16-bit and 32-bit CPUs
- x8-only Input/Output Architecture
  - 28F002BX-T 28F002BX-B
  - For Space Constrained 8-bit Applications
- Optimized High Density Blocked Architecture
  - One 16-KB Protected Boot Block
  - Two 8-KB Parameter Blocks
  - One 96-KB Main Block
  - One 128 KB Main Block
  - Top or Bottom Boot Locations
- Extended Cycling Capability
  - 100,000 Block Erase Cycles
- Automated Word/Byte Write and Block Erase
  - Command User Interface
  - Status Registers
  - Erase Suspend Capability
- SRAM-Compatible Write Interface
- Automatic Power Savings Feature
  - 1 mA Typical I_{CC} Active Current in Static Operation
- Hardware Data Protection Feature
  - Erase/Write Lockout during Power Transitions
- Very High-Performance Read
  - 60/80 ns Maximum Access Time
  - 30/40 ns Maximum Output Enable Time
- Low Power Consumption
  - 20 mA Typical x8 Active Read Current
  - 25 mA Typical x16 Active Read Current
- Reset/Deep Power-Down Input
  - 0.2 μA I_{CC} Typical
  - Acts as Reset for Boot Operations
- Extended Temperature Operation
  - $-40°C$ to $+85°C$
- Write Protection for Boot Block
- Industry Standard Surface Mount Packaging
  - 28F200BX: JEDEC ROM Compatible
    - 44-Lead PSOP
    - 56-Lead TSOP
  - 28F002BX: 40-Lead TSOP
- 12V Word/Byte Write and Block Erase
  - V_{PP} = 12V $\pm$5% Standard
  - V_{PP} = 12V $\pm$10% Option
- ETOX III Flash Technology
  - 5V Read
- Independent Software Vendor Support

Refer to the 1994 Memory Products Handbook for the complete data sheet on this device.

October 1993
Order Number: 290448-003

28F200BX-TL/BL, 28F002BX-TL/BL 2-MBIT (128K x 16, 256K x 8) LOW POWER BOOT BLOCK FLASH MEMORY FAMILY

- Low Voltage Operation for Very Low Power Portable Applications
  - V_{CC} = 3.3V ±0.3V
- x8/x16 Input/Output Architecture
  - 28F200BX-TL, 28F200BX-BL
  - For High Performance and High Integration 16-bit and 32-bit CPUs
- x8-only Input/Output Architecture
  - 28F002BX-TL, 28F002BX-BL
  - For Space Constrained 8-bit Applications
- Optimized High Density Blocked Architecture
  - One 16-KB Protected Boot Block
  - Two 8-KB Parameter Blocks
  - One 96-KB Main Block
  - One 128-KB Main Block
  - Top or Bottom Boot Locations
- Extended Cycling Capability
  - 10,000 Block Erase Cycles
- Automated Word/Byte Write and Block Erase
  - Command User Interface
  - Status Registers
  - Erase Suspend Capability
- SRAM-Compatible Write Interface
- Automatic Power Savings Feature
  - 0.8 mA Typical I_{CC} Active Current in Static Operation
- Very High-Performance Read
  - 150 ns Maximum Access Time
  - 65 ns Maximum Output Enable Time
- Low Power Consumption
  - 15 mA Typical Active Read Current
- Reset/Deep Power-Down Input
  - 0.2 µA I_{CC} Typical
  - Acts as Reset for Boot Operations
- Write Protection for Boot Block
- Hardware Data Protection Feature
  - Erase/Write Lockout during Power Transitions
- Industry Standard Surface Mount Packaging
  - 28F200BX-L: JEDEC ROM Compatible
    - 44-Lead PSOP
    - 56-Lead TSOP
  - 28F002BX-L: 40-Lead TSOP
- 12V Word/Byte Write and Block Erase
  - V_{PP} = 12V ±5% Standard
- ETOX III Flash Technology
  - 3.3V Read
- Independent Software Vendor Support

5

Refer to the 1994 Memory Products Handbook for the complete data sheet on this device.

October 1993
Order Number: 290449-003

ADVANCE INFORMATION

28F400BX-T/B, 28F004BX-T/B
4 MBIT (256K x16, 512K x8) BOOT BLOCK FLASH MEMORY FAMILY

- x8/x16 Input/Output Architecture
  - 28F400BX-T, 28F400BX-B
  - For High Performance and High Integration 16-bit and 32-bit CPUs
- x8-only Input/Output Architecture
  - 28F004BX-T, 28F004BX-B
  - For Space Constrained 8-bit Applications
- Optimized High Density Blocked Architecture
  - One 16-KB Protected Boot Block
  - Two 8-KB Parameter Blocks
  - One 96-KB Main Block
  - Three 128-KB Main Blocks
  - Top or Bottom Boot Locations
- Extended Cycling Capability
  - 100,000 Block Erase Cycles
- Automated Word/Byte Write and Block Erase
  - Command User Interface
  - Status Registers
  - Erase Suspend Capability
- SRAM-Compatible Write Interface
- Automatic Power Savings Feature
  - 1 mA Typical I_{CC} Active Current in Static Operation
- Very High-Performance Read
  - 60/80 ns Maximum Access Time
  - 30/40 ns Maximum Output Enable Time
- Low Power Consumption
  - 20 mA Typical x8 Active Read Current
  - 25 mA Typical x16 Active Read Current
- Reset/Deep Power-Down Input
  - 0.2 μA I_{CC} Typical
  - Acts as Reset for Boot Operations
- Extended Temperature Operation
  - $-40°C$ to $+85°C$
- Write Protection for Boot Block
- Hardware Data Protection Feature
  - Erase/Write Lockout During Power Transitions
- Industry Standard Surface Mount Packaging
  - 28F400BX: JEDEC ROM Compatible
    - 44-Lead PSOP
    - 56-Lead TSOP
  - 28F004BX: 40-Lead TSOP
- 12V Word/Byte Write and Block Erase
  - $V_{PP} = 12V \pm 5\%$ Standard
  - $V_{PP} = 12V \pm 10\%$ Option
- ETOX III Flash Technology
  - 5V Read

Refer to the 1994 Memory Products Handbook for the complete data sheet on this device.

October 1993
Order Number: 290451-003

28F008SA
8-MBIT (1-MBIT x 8) FLASH MEMORY

- **High-Density Symmetrically Blocked Architecture**
  - **Sixteen 64-Kbyte Blocks**
- **Extended Cycling Capability**
  - **100,000 Block Erase Cycles**
  - **1.6 Million Block Erase Cycles per Chip**
- **Automated Byte Write and Block Erase**
  - **Command User Interface**
  - **Status Register**
- **System Performance Enhancements**
  - **RY/BY# Status Output**
  - **Erase Suspend Capability**
- **Deep-Powerdown Mode**
  - **0.20 μA I_{CC} Typical**
- **Very High-Performance Read**
  - **85 ns Maximum Access Time**
- **SRAM-Compatible Write Interface**
- **Hardware Data Protection Feature**
  - **Erase/Write Lockout during Power Transitions**
- **Industry Standard Packaging**
  - **40-Lead TSOP, 44-Lead PSOP**
- **ETOX III Nonvolatile Flash Technology**
  - **12V Byte Write/Block Erase**
- **Independent Software Vendor Support**
  - **Microsoft* Flash File System (FFS)**

Intel's 28F008SA 8-Mbit FlashFile™ Memory is the highest density nonvolatile read/write solution for solid state storage. The 28F008SA's extended cycling, symmetrically blocked architecture, fast access time, write automation and low power consumption provide a more reliable, lower power, lighter weight and higher performance alternative to traditional rotating disk technology. The 28F008SA brings new capabilities to portable computing. Application and operating system software stored in resident flash memory arrays provide instant-on, rapid execute-in-place and protection from obsolescence through in-system software updates. Resident software also extends system battery life and increases reliability by reducing disk drive accesses.

For high density data acquisition applications, the 28F008SA offers a more cost-effective and reliable alternative to SRAM and battery. Traditional high density embedded applications, such as telecommunications, can take advantage of the 28F008SA's nonvolatility, blocking and minimal system code requirements for flexible firmware and modular software designs.

The 28F008SA is offered in 40-lead TSOP (standard and reverse) and 44-lead PSOP packages. Pin assignments simplify board layout when integrating multiple devices in a flash memory array or subsystem. This device uses an integrated Command User Interface and state machine for simplified block erasure and byte write. The 28F008SA memory map consists of 16 separately erasable 64-Kbyte blocks.

Intel's 28F008SA employs advanced CMOS circuitry for systems requiring low power consumption and noise immunity. Its 85 ns access time provides superior performance when compared with magnetic storage media. A deep powerdown mode lowers power consumption to 1 μW typical thru V_{CC}, crucial in portable computing, handheld instrumentation and other low-power applications. The RP# power control input also provides absolute data protection during system powerup/down.

Manufactured on Intel's 0.8 micron ETOX process, the 28F008SA provides the highest levels of quality, reliability and cost-effectiveness.

---

*Microsoft is a trademark of Microsoft Corporation.

Refer to the 1994 Memory Products Handbook for the complete data sheet on this device.

October 1993
Order Number: 290429-004

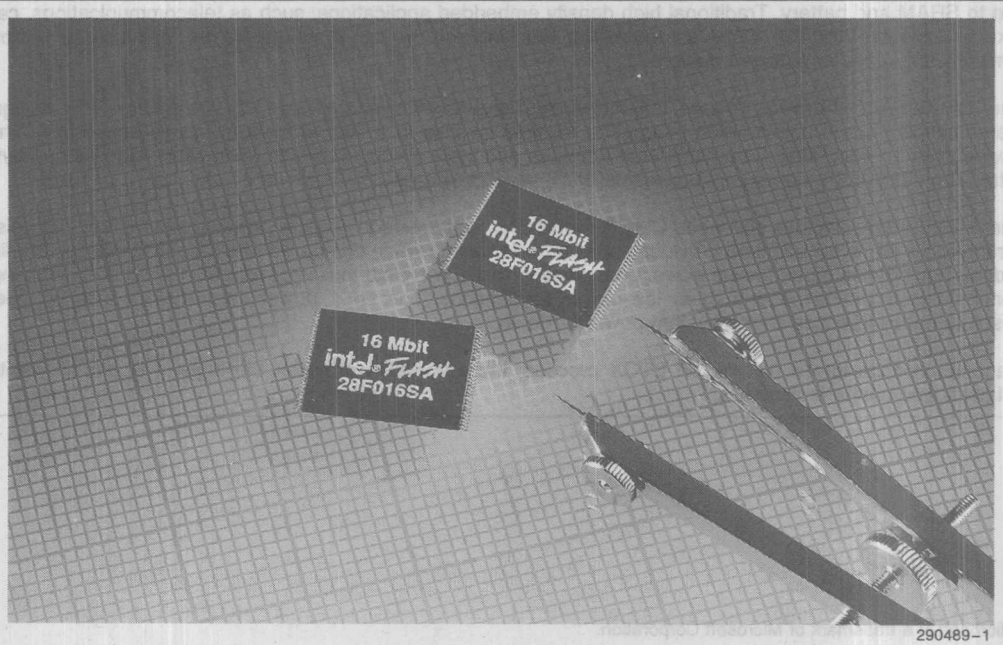
ADVANCE INFORMATION

28F016SA
16 MBIT (1 MBIT x 16, 2 MBIT x 8) FLASHFILE™ MEMORY

- **User-Selectable 3.3V or 5V V_{CC}**
- **User-Configurable x8 or x16 Operation**
- **70 ns Maximum Access Time**
- **0.43 MB/sec Write Transfer Rate**
- **1 Million Erase Cycles per Block**
- **56-Lead, 1.2mm x 14mm x 20mm TSOP Package**
- **Revolutionary Architecture**
  - **— Pipelined Command Execution**
  - **— Write During Erase**
  - **— Command Superset of Intel 28F008SA**
- **1 mA Typical I_{CC} in Static Mode**
- **1 μA Typical Deep Power-Down**
- **32 Independently Lockable Blocks**
- **State-of-the-Art 0.6 μm ETOX IV Flash Technology**

Intel's 28F016SA 16-Mbit FlashFile™ Memory is a revolutionary architecture which enables the design of truly mobile, high performance, personal computing and communication products. With innovative capabilities, low power operation and very high read/write performance, the 28F016SA is also the ideal choice for designing embedded mass storage flash memory systems.

The 28F016SA is a very high density, highest performance non-volatile read/write solution for solid-state storage applications. Its symmetrically blocked architecture (100% compatible with the 28F008SA 8-Mbit FlashFile™ memory), extended cycling, low power 3.3V operation, very fast write and read performance and selective block locking provide a highly flexible memory component suitable for high density memory cards, Resident Flash Arrays and PCMCIA-ATA Flash Drives. The 28F016SA's dual read voltage enables the design of memory cards which can interchangeably be read/written in 3.3V and 5.0V systems. Its x8/x16 architecture allows the optimization of memory to processor interface. The flexible block locking option enables bundling of executable application software in a Resident Flash Array or memory card. Manufactured on Intel's 0.6 μm ETOXIV process technology, the 28F016SA is the most cost-effective, high-density 3.3V flashfile memory.



290489-1

Refer to the 1994 Memory Products Handbook for the complete data sheet on this device.

October 1993
Order Number: 290489-001